Encyclopedia of Algorithms

Ming-Yang Kao (Ed.)

Encyclopedia of Algorithms

With 183 Figures and 38 Tables
With 4075 References for Further Reading

Springer

MING-YANG KAO
Professor of Computer Science
Department of Electrical Engineering and Computer Science
McCormick School of Engineering and Applied Science
Northwestern University
Evanston, IL 60208
USA

Library of Congress Control Number: 2007933824

ISBN: 978-0-387-30162-4

This publication is available also as:
Print publication under ISBN: 978-0-387-30770-1 and
Print and electronic bundle under ISBN: 978-0-387-36061-4

springer.com

Printed on acid free paper SPIN: 11563624 2109letex – 5 4 3 2 1 0

Preface

The *Encyclopedia of Algorithms* aims to provide the researchers, students, and practitioners of algorithmic research with a mechanism to efficiently and accurately find the names, definitions, key results, and further readings of important algorithmic problems.

The work covers a wide range of algorithmic areas, and each algorithmic area is covered by a collection of entries. An encyclopedia entry is an in-depth mini-survey of an algorithmic problem and is written by an expert researcher. The entries for an algorithmic area are compiled by an area editor to survey the representative results in that area and can form the core materials of a course in the area.

The *Encyclopedia* does not use the format of a conventional long survey for several reasons. A conventional survey takes a handful of individuals too much time to write and is difficult to update. An encyclopedia entry contains the same kinds of information as in a conventional survey, but an encyclopedia entry is much shorter and is much easier for readers to absorb and for editors to update. Furthermore, an algorithmic area is surveyed by a collection of entries which together provide a considerable amount of up-to-date information about the area, while the writing and updating of the entries is distributed among multiple authors to speed up the work.

This reference work will be updated on a regular basis and will evolve towards primarily an Internet-based medium to allow timely updates and fast search. If you have feedback regarding a particular entry, please feel free to communicate directly with the author or the area editor of that entry. If you are interested in authoring an entry, please contact a suitable area editor. If you have suggestions on how to improve the Encyclopedia as a whole, please contact me at kao@northwestern.edu.

The credit of the Encyclopedia goes to the area editors, the entry authors, the entry reviewers, and the project editors at Springer, including Jennifer Evans and Jennifer Carlson.

Ming-Yang Kao
Editor-in-Chief

Table of Contents

About the Editor

Ming-Yang Kao is a Professor of Computer Science in the Department of Electrical Engineering and Computer Science at Northwestern University. He has published extensively in the design, analysis, and applications of algorithms. His current interests include discrete optimization, bioinformatics, computational economics, computational finance, and nanotechnology. He serves as the Editor-in-Chief of Algorithmica.

He obtained a B.S. in Mathematics from National Taiwan University in 1978 and a Ph.D. in Computer Science from Yale University in 1986. He previously taught at Indiana University at Bloomington, Duke University, Yale University, and Tufts University. At Northwestern University, he has served as the Department Chair of Computer Science. He has also co-founded the Program in Computational Biology and Bioinformatics and served as its Director. He currently serves as the Head of the EECS Division of Computing, Algorithms, and Applications and is a member of the Theoretical Computer Science Group.

For more information please see: www.cs.northwestern.edu/~kao

Area Editors

Online Algorithms
Approximation Algorithms

ALBERS, SUSANNE
University of Freiburg
Freiburg
Germany

External Memory Algorithms and Data Structures
Cache-Oblivious Algorithms and Data Structures

ARGE, LARS
University of Aarhus
Aarhus
Denmark

Quantum Computing

© University of Latvia
Press Center

AMBAINIS, ANDRIS
University of Latvia
Riga
Latvia

Mechanism Design
Online Algorithms
Price of Anarchy

AZAR, YOSSI
Tel-Aviv University
Tel-Aviv
Israel

Approximation Algorithms

CHEKURI, CHANDRA
University of Illinois, Urbana-Champaign
Urbana, IL
USA

Online Algorithms
Radio Networks

CHROBAK, MAREK
University of California, Riverside
Riverside, CA
USA

Internet Algorithms
Network and Communication Protocols

COHEN, EDITH
AT&T Labs
Florham Park, NJ
USA

Bioinformatics

CSÜRÖS, MIKLÓS
University of Montreal
Montreal, QC
Canada

Computational Economics

DENG, XIAOTIE
University of Hong Kong
Hong Kong
China

Combinatorial Group Testing
Mathematical Optimization
Steiner Tree Algorithms

DU, DING-ZHU
University of Texas, Dallas
Richardson, TX
USA

String Algorithms and Data Structures
Data Compression

Ferragina, Paolo
University of Pisa
Pisa
Italy

Coding Algorithms

Guruswami, Venkatesan
University of Washington
Seattle, WA
USA

Algorithm Engineering
Dynamic Graph Algorithms

Italiano, Giuseppe
University of Rome
Rome
Italy

Stable Marriage Problems
Exact Algorithms

Iwama, Kazuo
Kyoto University
Kyoto
Japan

Approximation Algorithms

Khanna, Sanjeev
University of Pennsylvania
Philadelphia, PA
USA

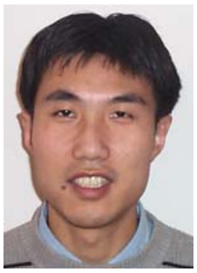
Graph Algorithms
Combinatorial Optimization
Approximation Algorithms

Khuller, Samir
University of Maryland
College Park, MD
USA

Compressed Text Indexing
Computational Biology

Lam, Tak-Wak
University of Hong Kong
Hong Kong
China

Mobile Computing



Li, Xiang-Yang
Illinois Institute of Technology
Chicago, IL
USA

Geometric Networks

Lingas, Andrzej
Lund University
Lund
Sweden

String Algorithms and Data Structures
Compression of Text
Data Structures

Navarro, Gonzalo
University of Chile
Santiago
Chile

Parameterized and Exact Algorithms

Neidermeier, Rolf
University of Jena
Jena
Germany

Probabilistic Algorithms
Average Case Analysis

Nikoletseas, Sotiris
Patras University
Patras
Greece

Graph Algorithms

PETTIE, SETH
University of Michigan
Ann Arbor, MI
USA

Scheduling Algorithms

PRUHS, KIRK
University of Pittsburgh
Pittsburgh, PA
USA

Distributed Algorithms

RAJSBAUM, SERGIO
National Autonomous University of Mexico
Mexico City
Mexico

Graph Algorithms

RAMACHANDRAN, VIJAYA
University of Texas, Austin
Austin, TX
USA

Algorithm Engineering

RAMAN, RAJEEV
University of Leicester
Leicester
UK

Computational Learning Theory

SERVEDIO, ROCCO
Columbia University
New York, NY
USA

Probabilistic Algorithms
Average Case Analysis

SPIRAKIS, PAVLOS (PAUL)
Patras University
Patras
Greece

Scheduling Algorithms

STEIN, CLIFFORD
Columbia University
New York, NY
USA

VLSI CAD Algorithms

ZHOU, HAI
Northwestern University
Evanston, IL
USA

List of Contributors

AARDAL, KAREN
CWI
Amsterdam
The Netherlands
Eindhoven University of Technology
Eindhoven
The Netherlands

AKAVIA, ADI
MIT
Cambridge, MA
USA

ALBERS, SUSANNE
University of Freiburg
Freiburg
Germany

ALICHERRY, MANSOOR
Bell Labs
Murray Hill, NJ
USA

ALON, NOGA
Tel-Aviv University
Tel-Aviv
Israel

ALTSCHUL, STEPHEN F.
The Rockefeller University
New York, NY
USA
MIT
Cambridge, MA
USA

ALURU, SRINIVAS
Iowa State University
Ames, IA
USA

AMBAINIS, ANDRIS
University of Latvia
Riga
Latvia

AMBÜHL, CHRISTOPH
University of Liverpool
Liverpool
UK

AMIR, AMIHOOD
Bar-Ilan University
Ramat-Gan
Israel

ASODI, VERA
California Institute of Technology
Pasadena, CA
USA

AUER, PETER
University of Leoben
Leoben
Austria

AZIZ, ADNAN
University of Texas
Austin, TX
USA

BABAIOFF, MOSHE
Microsoft Research, Silicon Valley
Mountain View, CA
USA

BADER, DAVID A.
Georgia Institute of Technology
Atlanta, GA
USA

BAEZA-YATES, RICARDO
University of Chile
Santiago
Chile

BANSAL, NIKHIL
IBM
Yorktown Heights, NY
USA

BARBAY, JÉRÉMY
University of Chile
Santiago
Chile

BARUAH, SANJOY
University of North Carolina
Chapel Hill, NC
USA

BASWANA, SURENDER
IIT Kanpur
Kanpur
India

BECCHETTI, LUCA
University of Rome
Rome
Italy

BEIMEL, AMOS
Ben-Gurion University
Beer Sheva
Israel

BÉKÉSI, JÓZSEF
Juhász Gyula Teachers Training College
Szeged
Hungary

BERGADANO, FRANCESCO
University of Torino
Torino
Italy

BERRY, VINCENT
LIRMM, University of Montpellier
Montpellier
France

BHATIA, RANDEEP
Bell Labs
Murray Hill, NJ
USA

BJÖRKLUND, ANDREAS
Lund University
Lund
Sweden

BLANCHETTE, MATHIEU
McGill University
Montreal, QC
Canada

BLÄSER, MARKUS
Saarland University
Saarbrücken
Germany

BODLAENDER, HANS L.
University of Utrecht
Utrecht
The Netherlands

BORRADAILE, GLENCORA
Brown University
Providence, RI
USA

BSHOUTY, NADER H.
Technion
Haifa
Israel

BUCHSBAUM, ADAM L.
AT&T Labs, Inc.
Florham Park, NJ
USA

BUSCH, COSTAS
Lousiana State University
Baton Rouge, LA
USA

BU, TIAN-MING
Fudan University
Shanghai
China

BYRKA, JAROSLAW
CWI
Amsterdam
The Netherlands
Eindhoven University of Technology
Eindhoven
The Netherlands

CAI, MAO-CHENG
Chinese Academy of Sciences
Beijing
China

CALINESCU, GRUIA
Illinois Institute of Technology
Chicago, IL
USA

CECHLÁROVÁ, KATARÍNA
P.J. Šafárik University
Košice
Slovakia

CHAN, CHEE-YONG
National University of Singapore
Singapore
Singapore

CHANDRA, TUSHAR DEEPAK
IBM Watson Research Center
Yorktown Heights, NY
USA

CHAO, KUN-MAO
National Taiwan University
Taipei
Taiwan

CHARRON-BOST, BERNADETTE
The Polytechnic School
Palaiseau
France

CHATZIGIANNAKIS, IOANNIS
University of Patras and Computer Technology Institute
Patras
Greece

CHAWLA, SHUCHI
University of Wisconsin–Madison
Madison, WI
USA

CHEKURI, CHANDRA
University of Illinois, Urbana-Champaign
Urbana, IL
USA

CHEN, DANNY Z.
University of Notre Dame
Notre Dame, IN
USA

CHENG, XIUZHEN
The George Washington University
Washington, D.C.
USA

CHEN, JIANER
Texas A&M University
College Station, TX
USA

CHEN, XI
Tsinghua University
Beijing, Beijing
China

CHIN, FRANCIS
University of Hong Kong
Hong Kong
China

CHOWDHURY, REZAUL A.
University of Texas at Austin
Austin, TX
USA

CHRISTODOULOU, GEORGE
Max-Planck-Institute for Computer Science
Saarbruecken
Germany

CHROBAK, MAREK
University of California at Riverside
Riverside, CA
USA

CHU, CHRIS
Iowa State University
Ames, IA
USA

CHU, XIAOWEN
Hong Kong Baptist University
Hong Kong
China

CHUZHOY, JULIA
Toyota Technological Institute
Chicago, IL
USA

CONG, JASON
UCLA
Los Angeles, CA
USA

COWEN, LENORE J.
Tufts University
Medford, MA
USA

CRISTIANINI, NELLO
University of Bristol
Bristol
UK

CROCHEMORE, MAXIME
King's College London
London
UK
University of Paris-East
Paris
France

CSŰRÖS, MIKLÓS
University of Montreal
Montreal, QC
Canada

CZUMAJ, ARTUR
University of Warwick
Coventry
UK

DASGUPTA, BHASKAR
University of Illinois at Chicago
Chicago, IL
USA

DÉFAGO, XAVIER
Japan Advanced Institute of Science and Technology (JAIST)
Ishikawa
Japan

DEMAINE, ERIK D.
MIT
Cambridge, MA
USA

DEMETRESCU, CAMIL
University of Rome
Rome
Italy

DENG, PING
University of Texas at Dallas
Richardson, TX
USA

DENG, XIAOTIE
City University of Hong Kong
Hong Kong
China

DESPER, RICHARD
University College London
London
UK

DICK, ROBERT
Northwestern University
Evanston, IL
USA

DING, YUZHENG
Synopsys Inc.
Mountain View, CA
USA

DOM, MICHAEL
University of Jena
Jena
Germany

DUBHASHI, DEVDATT
Chalmers University of Technology and Gothenburg University
Gothenburg
Sweden

DU, DING-ZHU
University of Dallas at Texas
Richardson, TX
USA

EDMONDS, JEFF
York University
Toronto, ON
Canada

EFRAIMIDIS, PAVLOS
Democritus University of Thrace
Xanthi
Greece

EFTHYMIOU, CHARILAOS
University of Patras
Patras
Greece

ELKIN, MICHAEL
Ben-Gurion University
Beer-Sheva
Israel

EPSTEIN, LEAH
University of Haifa
Haifa
Israel

ERICKSON, BRUCE W.
The Rockefeller University
New York, NY
USA

EVEN-DAR, EYAL
University of Pennsylvania
Philadelphia, PA
USA

FAGERBERG, ROLF
University of Southern Denmark
Odense
Denmark

FAKCHAROENPHOL, JITTAT
Kasetsart University
Bangkok
Thailand

FANG, QIZHI
Ocean University of China
Qingdao
China

FATOUROU, PANAGIOTA
University of Ioannina
Ioannina
Greece

FELDMAN, JONATHAN
Google, Inc.
New York, NY
USA

FELDMAN, VITALY
Harvard University
Cambridge, MA
USA

FERNAU, HENNING
University of Trier
Trier
Germany

FERRAGINA, PAOLO
University of Pisa
Pisa
Italy

FEUERSTEIN, ESTEBAN
University of Buenos Aires
Buenos Aires
Argentina

FISHER, NATHAN
University of North Carolina
Chapel Hill, NC
USA

FLAXMAN, ABRAHAM
Microsoft Research
Redmond, WA
USA

FLEISCHER, RUDOLF
Fudan University
Shanghai
China

FOMIN, FEDOR
University of Bergen
Bergen
Norway

FOTAKIS, DIMITRIS
University of the Aegean
Samos
Greece

FRIEDER, OPHIR
Illinois Institute of Technology
Chicago, IL
USA

FÜRER, MARTIN
The Pennsylvania State University
University Park, PA
USA

GAGIE, TRAVIS
University of Eastern Piedmont
Alessandria
Italy

GALAMBOS, GÁBOR
Juhász Gyula Teachers Training College
Szeged
Hungary

GAO, JIE
Stony Brook University
Stony Brook, NY
USA

GARAY, JUAN
Bell Labs
Murray Hill, NJ
USA

GAROFALAKIS, MINOS
University of California – Berkeley
Berkeley, CA
USA

GASCUEL, OLIVIER
National Scientific Research Center
Montpellier
France

GĄSIENIEC, LESZEK
University of Liverpool
Liverpool
UK

GIANCARLO, RAFFAELE
University of Palermo
Palermo
Italy

GOLDBERG, ANDREW V.
Microsoft Research – Silicon Valley
Mountain View, CA
USA

GRAMM, JENS
Tübingen University
Tübingen
Germany

GROVER, LOV K.
Bell Labs
Murray Hill, NJ
USA

GUDMUNDSSON, JOACHIM
National ICT Australia Ltd
Alexandria
Australia

GUERRAOUI, RACHID
EPFL
Lausanne
Switzerland

GUO, JIONG
University of Jena
Jena
Germany

GURUSWAMI, VENKATESAN
University of Washington
Seattle, WA
USA

HAJIAGHAYI, MOHAMMADTAGHI
University of Pittsburgh
Pittsburgh, PA
USA

HALLGREN, SEAN
The Pennsylvania State University
University Park, PA
USA

HALPERIN, DAN
Tel-Aviv University
Tel Aviv
Israel

HARIHARAN, RAMESH
Strand Life Sciences
Bangalore
India

HELLERSTEIN, LISA
Polytechnic University
Brooklyn, NY
USA

HE, MENG
University of Waterloo
Waterloo, ON
Canada

HENZINGER, MONIKA
Google Switzerland & Ecole Polytechnique Federale de Lausanne (EPFL)
Lausanne
Switzerland

HERLIHY, MAURICE
Brown University
Providence, RI
USA

HERMAN, TED
University of Iowa
Iowa City, IA
USA

HE, XIN
University at Buffalo The State University of New York
Buffalo, NY
USA

HIRSCH, EDWARD A.
Steklov Institute of Mathematics at St. Petersburg
St. Petersburg
Russia

HON, WING-KAI
National Tsing Hua University
Hsin Chu
Taiwan

HOWARD, PAUL G.
Microway, Inc.
Plymouth, MA
USA

HUANG, LI-SHA
Tsinghua University
Beijing, Beijing
China

HUANG, YAOCUN
University of Texas at Dallas
Richardson, TX
USA

HÜFFNER, FALK
University of Jena
Jena
Germany

HUSFELDT, THORE
Lund University
Lund
Sweden

ILIE, LUCIAN
University of Western Ontario
London, ON
Canada

IRVING, ROBERT W.
University of Glasgow
Glasgow
UK

ITAI, ALON
Technion
Haifa
Israel

ITALIANO, GIUSEPPE F.
University of Rome
Rome
Italy

IWAMA, KAZUO
Kyoto University
Kyoto
Japan

JACKSON, JEFFREY C.
Duquesne University
Pittsburgh, PA
USA

JACOB, RIKO
Technical University of Munich
Munich
Germany

JAIN, RAHUL
University of Waterloo
Waterloo, ON
Canada

JANSSON, JESPER
Ochanomizu University
Tokyo
Japan

JIANG, TAO
University of California at Riverside
Riverside, CA
USA

JOHNSON, DAVID S.
AT&T Labs
Florham Park, NJ
USA

KAJITANI, YOJI
The University of Kitakyushu
Kitakyushu
Japan

KAPORIS, ALEXIS
University of Patras
Patras
Greece

KARAKOSTAS, GEORGE
McMaster University
Hamilton, ON
Canada

KÄRKKÄINEN, JUHA
University of Helsinki
Helsinki
Finland

KELLERER, HANS
University of Graz
Graz
Austria

KENNINGS, ANDREW A.
University of Waterloo
Waterloo, ON
Canada

KEUTZER, KURT
University of California at Berkeley
Berkeley, CA
USA

KHULLER, SAMIR
University of Maryland
College Park, MD
USA

KIM, JIN WOOK
HM Research
Seoul
Korea

KIM, YOO-AH
University of Connecticut
Storrs, CT
USA

KING, VALERIE
University of Victoria
Victoria, BC
Canada

KIROUSIS, LEFTERIS
University of Patras
Patras
Greece

KIVINEN, JYRKI
University of Helsinki
Helsinki
Finland

KLEIN, ROLF
University of Bonn
Bonn
Germany

KLIVANS, ADAM
University of Texas at Austin
Austin, TX
USA

KONJEVOD, GORAN
Arizona State University
Tempe, AZ
USA

KONTOGIANNIS, SPYROS
University of Ioannina
Ioannina
Greece

KRANAKIS, EVANGELOS
Carleton
Ottawa, ON
Canada

KRATSCH, DIETER
Paul Verlaine University
Metz
France

KRAUTHGAMER, ROBERT
Weizmann Institute of Science
Rehovot
Israel
IBM Almaden Research Center
San Jose, CA
USA

KRIZANC, DANNY
Wesleyan University
Middletown, CT
USA

KRYSTA, PIOTR
University of Liverpool
Liverpool
UK

KUCHEROV, GREGORY
LIFL and INRIA
Villeneuve d'Ascq
France

KUHN, FABIAN
ETH Zurich
Zurich
Switzerland

KUMAR, V.S. ANIL
Virginia Tech
Blacksburg, VA
USA

KUSHILEVITZ, EYAL
Technion
Haifa
Israel

LAM, TAK-WAH
University of Hong Kong
Hong Kong
China

LANCIA, GIUSEPPE
University of Udine
Udine
Italy

LANDAU, GAD M.
University of Haifa
Haifa
Israel

LANDAU, ZEPH
City College of CUNY
New York, NY
USA

LANGBERG, MICHAEL
The Open University of Israel
Raanana
Israel

LAVI, RON
Technion
Haifa
Israel

LECROQ, THIERRY
University of Rouen
Rouen
France

LEE, JAMES R.
University of Washington
Seattle, WA
USA

LEONARDI, STEFANO
University of Rome
Rome
Italy

LEONE, PIERRE
University of Geneva
Geneva
Switzerland

LEUNG, HENRY
MIT
Cambridge, MA
USA

LEVCOPOULOS, CHRISTOS
Lund University
Lund
Sweden

LEWENSTEIN, MOSHE
Bar-Ilan University
Ramat-Gan
Israel

LI, LI (ERRAN)
Bell Labs
Murray Hill, NJ
USA

LI, MING
University of Waterloo
Waterloo, ON
Canada

LI, MINMING
City University of Hong Kong
Hong Kong
China

LINGAS, ANDRZEJ
Lund University
Lund
Sweden

LI, XIANG-YANG
Illinois Institue of Technology
Chicago, IL
USA

LU, CHIN LUNG
National Chiao Tung University
Hsinchu
Taiwan

LYNGSØ, RUNE B.
Oxford University
Oxford
UK

MA, BIN
University of Western Ontario
London, ON
Canada

MAHDIAN, MOHAMMAD
Yahoo! Research
Santa Clara, CA
USA

MÄKINEN, VELI
University of Helsinki
Helsinki
Finland

MALKHI, DAHLIA
Microsoft, Silicon Valley Campus
Mountain View, CA
USA

MANASSE, MARK S.
Microsoft Research
Mountain View, CA
USA

MANLOVE, DAVID F.
University of Glasgow
Glasgow
UK

MANZINI, GIOVANNI
University of Eastern Piedmont
Alessandria
Italy

MARATHE, MADHAV V.
Virginia Tech
Blacksburg, VA
USA

MARCHETTI-SPACCAMELA, ALBERTO
University of Rome
Rome
Italy

MARKOV, IGOR L.
University of Michigan
Ann Arbor, MI
USA

MCGEOCH, CATHERINE C.
Amherst College
Amherst, MA
USA

MCGEOCH, LYLE A.
Amherst College
Amherst, MA
USA

MCKAY, BRENDAN D.
Australian National University
Canberra, ACT
Australia

MENDEL, MANOR
The Open University of Israel
Raanana
Israel

MESTRE, JULIÁN
University of Maryland
College Park, MD
USA

MICCIANCIO, DANIELE
University of California, San Diego
La Jolla, CA
USA

MIKLÓS, ISTVÁN
Eötvös Lóránd University
Budapest
Hungary

MIRROKNI, VAHAB S.
Microsoft Research
Redmond, WA
USA

MIYAZAKI, SHUICHI
Kyoto University
Kyoto
Japan

MOFFAT, ALISTAIR
University of Melbourne
Melbourne, VIC
Australia

MOIR, MARK
Sun Microsystems Laboratories
Burlington, MA
USA

MOR, TAL
Technion
Haifa
Israel

MOSCA, MICHELE
University of Waterloo
Waterloo, ON
Canada
St. Jerome's University
Waterloo, ON
Canada

MOSCIBRODA, THOMAS
Microsoft Research
Redmond, WA
USA

MUCHA, MARCIN
Institute of Informatics
Warsaw
Poland

MUNAGALA, KAMESH
Duke University
Durham, NC
USA

MUNRO, J. IAN
University of Waterloo
Waterloo, ON
Canada

NA, JOONG CHAE
Sejong University
Seoul
Korea

NARASIMHAN, GIRI
Florida International University
Miami, FL
USA

NAVARRO, GONZALO
University of Chile
Santiago
Chile

NAYAK, ASHWIN
University of Waterloo and Perimeter Institute for Theoretical Physics
Waterloo, ON
Canada

NEWMAN, ALANTHA
Max-Planck Institute for Computer Science
Saarbrücken
Germany

NIEDERMEIER, ROLF
University of Jena
Jena
Germany

NIKOLETSEAS, SOTIRIS
University of Patras
Patras
Greece

OKAMOTO, YOSHIO
Toyohashi University of Technology
Toyohashi
Japan

OKUN, MICHAEL
Weizmann Institute of Science
Rehovot
Israel

PAGH, RASMUS
IT University of Copenhagen
Copenhagen
Denmark

PANAGOPOULOU, PANAGIOTA
Research Academic Computer Technology Institute
Patras
Greece

PANIGRAHI, DEBMALYA
MIT
Cambridge, MA
USA

PAN, PEICHEN
Magma Design Automation, Inc.
Los Angeles, CA
USA

PAPADOPOULOU, VICKY
University of Cyprus
Nicosia
Cyprus

PARK, KUNSOO
Seoul National University
Seoul
Korea

PARTHASARATHY, SRINIVASAN
IBM T.J. Watson Research Center
Hawthorne, NY
USA

PĂTRAŞCU, MIHAI
MIT
Cambridge, MA
USA

PATT-SHAMIR, BOAZ
Tel-Aviv University
Tel-Aviv
Israel

PATURI, RAMAMOHAN
University of California at San Diego
San Diego, CA
USA

PELC, ANDRZEJ
University of Québec-Ottawa
Gatineau, QC
Canada

PETTIE, SETH
University of Michigan
Ann Arbor, MI
USA

POWELL, OLIVIER
University of Geneva
Geneva
Switzerland

PRAKASH, AMIT
Microsoft, MSN
Redmond, WA
USA

PRUHS, KIRK
University of Pittsburgh
Pittsburgh, PA
USA

PRZYTYCKA, TERESA M.
NIH
Bethesda, MD
USA

PUDLÁK, PAVEL
Academy of Science of the Czech Republic
Prague
Czech Republic

RAGHAVACHARI, BALAJI
University of Texas at Dallas
Richardson, TX
USA

RAHMAN, NAILA
University of Leicester
Leicester
UK

RAJARAMAN, RAJMOHAN
Northeastern University
Boston, MA
USA

RAJSBAUM, SERGIO
National Autonomous University of Mexico
Mexico City
Mexico

RAMACHANDRAN, VIJAYA
University of Texas at Austin
Austin, TX
USA

RAMAN, RAJEEV
University of Leicester
Leicester
UK

RAMOS, EDGAR
National University of Colombia
Medellín
Colombia

RAO, SATISH
University of California at Berkeley
Berkeley, CA
USA

RAO, S. SRINIVASA
IT University of Copenhagen
Copenhagen
Denmark

RAPTOPOULOS, CHRISTOFOROS
University of Patras
Patras
Greece

RASTOGI, RAJEEV
Lucent Technologies
Murray Hill, NJ
USA

RATSABY, JOEL
Ariel University Center of Samaria
Ariel
Israel

RAVINDRAN, KAUSHIK
University of California at Berkeley
Berkeley, CA
USA

RAYNAL, MICHEL
University of Rennes 1
Rennes
France

REICHARDT, BEN W.
California Institute of Technology
Pasadena, CA
USA

RENNER, RENATO
Institute for Theoretical Physics
Zurich
Switzerland

RICCI, ELISA
University of Perugia
Perugia
Italy

RICHTER, PETER
Rutgers, The State University of New Jersey
Piscataway, NJ
USA

ROLIM, JOSÉ
University of Geneva
Geneva
Switzerland

ROSAMOND, FRANCES
University of Newcastle
Callaghan, NSW
Australia

RÖTTELER, MARTIN
NEC Laboratories America
Princeton, NJ
USA

RUBINFELD, RONITT
MIT
Cambridge, MA
USA

RUDRA, ATRI
University at Buffalo, State University of New York
Buffalo, NY
USA

RUPPERT, ERIC
York University
Toronto, ON
Canada

RYTTER, WOJCIECH
Warsaw University
Warsaw
Poland

SAHINALP, S. CENK
Simon Fraser University
Burnaby, BC
USA

SAKS, MICHAEL
Rutgers, State University of New Jersey
Piscataway, NJ
USA

SCHÄFER, GUIDO
Technical University of Berlin
Berlin
Germany

SCHIPER, ANDRÉ
EPFL
Lausanne
Switzerland

SCHMIDT, MARKUS
University of Freiburg
Freiburg
Germany

SCHULTES, DOMINIK
University of Karlsruhe
Karlsruhe
Germany

SEN, PRANAB
Tata Institute of Fundamental Research
Mumbai
India

SEN, SANDEEP
IIT Delhi
New Delhi
India

SERNA, MARIA
Technical University of Catalonia
Barcelona
Spain

SERVEDIO, ROCCO
Columbia University
New York, NY
USA

SETHURAMAN, JAY
Columbia University
New York, NY
USA

SHALEV-SHWARTZ, SHAI
Toyota Technological Institute
Chicago, IL
USA

SHARMA, VIKRAM
New York University
New York, NY
USA

SHI, YAOYUN
University of Michigan
Ann Arbor, MI
USA

SHRAGOWITZ, EUGENE
University of Minnesota
Minneapolis, MN
USA

SITTERS, RENÉ A.
Eindhoven University of Technology
Eindhoven
The Netherlands

SMID, MICHIEL
Carleton University
Ottawa, ON
Canada

SOKOL, DINA
Brooklyn College of CUNY
Brooklyn, NY
USA

SONG, WEN-ZHAN
Washington State University
Vancouver, WA
USA

SPECKMANN, BETTINA
Technical University of Eindhoven
Eindhoven
The Netherlands

SPIRAKIS, PAUL
Patras University
Patras
Greece

SRINIVASAN, ARAVIND
University of Maryland
College Park, MD
USA

SRINIVASAN, VENKATESH
University of Victoria
Victoria, BC
Canada

STEE, ROB VAN
University of Karlsruhe
Karlsruhe
Germany

STØLTING BRODAL, GERTH
University of Aarhus
Århus
Denmark

STOYE, JENS
University of Bielefeld
Bielefeld
Germany

SU, CHANG
University of Liverpool
Liverpool
UK

SUN, ARIES WEI
City University of Hong Kong
Hong Kong
China

SUNDARARAJAN, VIJAY
Texas Instruments
Dallas, TX
USA

SUNG, WING-KIN
National University of Singapore
Singapore
Singapore

SVIRIDENKO, MAXIM
IBM
Yorktown Heights, NY
USA

SZEGEDY, MARIO
Rutgers, The State University of New Jersey
Piscataway, NJ
USA

SZEIDER, STEFAN
Durham University
Durham
UK

TAKAOKA, TADAO
University of Canterbury
Christchurch
New Zealand

TAKEDA, MASAYUKI
Kyushu University
Fukuoka
Japan

TALWAR, KUNAL
Microsoft Research, Silicon Valley Campus
Mountain View, CA
USA

TAMON, CHRISTINO
Clarkson University
Potsdam, NY
USA

TAMURA, AKIHISA
Keio University
Yokohama
Japan

TANNIER, ERIC
University of Lyon
Lyon
France

TAPP, ALAIN
University of Montréal
Montreal, QC
Canada

TATE, STEPHEN R.
University of North Carolina at Greensboro
Greensboro, NC
USA

TAUBENFELD, GADI
Interdiciplinary Center Herzlia
Herzliya
Israel

TELIKEPALLI, KAVITHA
Indian Institute of Science
Bangalore
India

TERHAL, BARBARA M.
IBM Research
Yorktown Heights, NY
USA

THILIKOS, DIMITRIOS
National and Kapodistrian University of Athens
Athens
Greece

TREVISAN, LUCA
University of California at Berkeley
Berkeley, CA
USA

TROMP, JOHN
CWI
Amsterdam
Netherlands

UKKONEN, ESKO
University of Helsinki
Helsinki
Finland

VAHRENHOLD, JAN
Dortmund University of Technology
Dortmund
Germany

VARRICCHIO, STEFANO
University of Roma
Rome
Italy

VIALETTE, STÉPHANE
University of Paris-East
Descartes
France

VILLANGER, YNGVE
University of Bergen
Bergen
Norway

VITÁNYI, PAUL
CWI
Amsterdam
Netherlands

VITTER, JEFFREY SCOTT
Purdue University
West Lafayette, IN
USA

VÖCKING, BERTHOLD
RWTH Aachen University
Aachen
Germany

WANG, CHENGWEN CHRIS
Carnegie Mellon University
Pittsburgh, PA
USA

WANG, FENG
Arizona State University
Phoenix, AZ
USA

WANG, LUSHENG
City University of Hong Kong
Hong Kong
China

WANG, WEIZHAO
Google Inc.
Irvine, CA
USA

WANG, YU
University of North Carolina at Charlotte
Charlotte, NC
USA

WAN, PENG-JUN
Illinois Institute of Technology
Chicago, IL
USA

WERNECK, RENATO F.
Microsoft Research Silicon Valley
La Avenida, CA
USA

WILLIAMS, RYAN
Carnegie Mellon University
Pittsburgh, PA
USA

WONG, MARTIN D. F.
University of Illinois at Urbana-Champaign
Urbana, IL
USA

WONG, PRUDENCE
University of Liverpool
Liverpool
UK

WU, WEILI
University of Texas at Dallas
Richardson, TX
USA

YANG, HONGHUA HANNAH
Intel Corporation
Hillsboro
USA

YAP, CHEE K.
New York University
New York, NY
USA

YE, YIN-YU
Stanford University
Stanford, CA
USA

YI, CHIH-WEI
National Chiao Tung University
Hsinchu City
Taiwan

YI, KE
Hong Kong University of Science and Technology
Hong Kong
China

YIU, S. M.
The University of Hong Kong
Hong Kong
China

YOKOO, MAKOTO
Kyushu University
Nishi-ku
Japan

YOUNG, EVANGELINE F. Y.
The Chinese University of Hong Kong
Hong Kong
China

YOUNG, NEAL E.
University of California at Riverside
Riverside, CA
USA

YUSTER, RAPHAEL
University of Haifa
Haifa
Israel

ZANE, FRANCIS
Lucent Technologies
Murray Hill, NJ
USA

ZAROLIAGIS, CHRISTOS
University of Patras
Patras
Greece

ZEH, NORBERT
Dalhousie University
Halifax, NS
Canada

ZHANG, LI
HP Labs
Palo Alto, CA
USA

ZHANG, LOUXIN
National University of Singapore
Singapore
Singapore

ZHOU, HAI
Northwestern University
Evanston, IL
USA

ZILLES, SANDRA
University of Alberta
Edmonton, AB
Canada

ZOLLINGER, AARON
University of California at Berkeley
Berkeley, CA
USA

ZWICK, URI
Tel-Aviv University
Tel-Aviv
Israel

A

Abelian Hidden Subgroup Problem

1995; Kitaev

MICHELE MOSCA[1,2]
[1] Combinatorics and Optimization / Institute for Quantum Computing, University of Waterloo, Waterloo, ON, Canada
[2] Perimeter Institute for Theoretical Physics, St. Jerome's University, Waterloo, ON, Canada

Keywords and Synonyms

Generalization of Abelian stabilizer problem; Generalization of Simon's problem

Problem Definition

The Abelian hidden subgroup problem is the problem of finding generators for a subgroup K of an Abelian group G, where this subgroup is defined implicitly by a function $f\colon G \to X$, for some finite set X. In particular, f has the property that $f(v) = f(w)$ if and only if the cosets[1] $v + K$ and $w + K$ are equal. In other words, f is constant on the cosets of the subgroup K, and distinct on each coset.

It is assumed that the group G is finitely generated and that the elements of G and X have unique binary encodings (the binary assumption is not so important, but it is important to have unique encodings.) When using variables g and h (possibly with subscripts) multiplicative notation is used for the group operations. Variables x and y (possibly with subscripts) will denote integers with addition. The boldface versions $\mathbf{x}$ and $\mathbf{y}$ will denote *tuples* of integers or binary strings.

By assumption, there is computational means of computing the function f, typically a circuit or "black box" that maps the encoding of a value g to the encoding of $f(g)$. The theory of reversible computation implies that one can turn a circuit for computing $f(g)$ into a reversible circuit for computing $f(g)$ with a modest increase in the size of the circuit. Thus it will be assumed that there is a reversible circuit or black box that maps $(g, \mathbf{z}) \mapsto (g, \mathbf{z} \oplus f(g))$, where $\oplus$ denotes the bitwise *XOR* (sum modulo 2), and $\mathbf{z}$ is any binary string of the same length as the encoding of $f(g)$.

Quantum mechanics implies that any reversible gate can be extended linearly to a unitary operation that can be implemented in the model of quantum computation. Thus, it is assumed that there is a quantum circuit or black box that implements the unitary map $U_f\colon |g\rangle|\mathbf{z}\rangle \mapsto |g\rangle|\mathbf{z} \oplus f(g)\rangle$.

Although special cases of this problem have been considered in classical computer science, the general formulation as the hidden subgroup problem seems to have appeared in the context of quantum computing, since it neatly encapsulates a family of "black-box" problems for which quantum algorithms offer an exponential speed up (in terms of query complexity) over classical algorithms. For some explicit problems (i. e., where the black box is replaced with a specific function, such as exponentiation modulo N), there is a conjectured exponential speed up.

Abelian Hidden Subgroup Problem

Input: Elements $g_1, g_2, \dots, g_n \in G$ that generate the Abelian group G. A black box that implements $U_f\colon |m_1, m_2, \dots, m_n\rangle|\mathbf{y}\rangle \mapsto |m_1, m_2, \dots, m_n\rangle|f(g) \oplus \mathbf{y}\rangle$, where $g = g_1^{m_1} g_2^{m_2} \dots g_n^{m_n}$, and K is the hidden subgroup corresponding to f.
Output: Elements $h_1, h_2, \dots, h_l \in G$ that generate K.

Here we use multiplicative notation for the group G in order to be consistent with Kitaev's formulation of the Abelian stabilizer problem. Many of the applications of interest typically use additive notation for the group G.

It is hard to trace the precise origin of this general formulation of the problem, which simultaneously general-

[1] Assuming additive notation for the group operation here.

izes "Simon's problem" [16], the order-finding problem (which is the quantum part of the quantum factoring algorithm [14]) and the discrete logarithm problem.

One of the earliest generalizations of Simon's problem, the order-finding problem, and the discrete logarithm problem, which captures the essence of the Abelian hidden subgroup problem is the *Abelian stabilizer problem*, which was solved by Kitaev [11] using a quantum algorithm in his 1995 paper (and the solution also appears in [12]).

Let G be a group acting on a finite set X. That is, each element of G acts as a map from X to X in such a way that for any two elements $g, h \in G$, $g(h(z)) = (gh)(z)$ for all $z \in X$. For a particular element $z \in X$, the set of elements that fix z (that is the elements $g \in G$ such that $g(z) = z$) form a subgroup. This subgroup is called the stabilizer of z in G, denoted $St_G(z)$.

Abelian Stabilizer Problem

Input: Elements $g_1, g_2, \ldots, g_n \in G$ that generate the group G. An element $z \in X$. A black box that implements $U_{(G,X)} : |m_1, m_2, \ldots, m_n\rangle|z\rangle \mapsto |m_1, m_2, \ldots, m_n\rangle|g(z)\rangle$ where $g = g_1^{m_1} g_2^{m_2} \ldots g_n^{m_n}$.
Output: Elements $h_1, h_2, \ldots, h_l \in G$ that generate $St_G(z)$.

Let f_z denote the function from G to X that maps $g \in G$ to $g(z)$. One can implement U_{f_z} using $U_{(G,X)}$. The hidden subgroup corresponding to f_z is $St_G(z)$. Thus, the Abelian stabilizer problem is a special case of the Abelian hidden subgroup problem.

One of the subtle differences (discussed in Appendix 6 of [10]) between the above formulation of the Abelian stabilizer problem and the Abelian hidden subgroup problem is that Kitaev's formulation gives a black box that for any $g, h \in G$ maps $|m_1, \ldots, m_n\rangle|f_z(g)\rangle \mapsto |m_1, \ldots, m_n\rangle|f_z(hg)\rangle$, where $g = g_1^{m_1} g_2^{m_2} \ldots g_n^{m_n}$ and estimates eigenvalues of shift operations of the form $|f_z(g)\rangle \mapsto |f_z(hg)\rangle$. In general, these shift operators are not explicitly needed, and it suffices to be able to compute a map of the form $|\mathbf{y}\rangle \mapsto |f_z(h) \oplus \mathbf{y}\rangle$ for any binary string $\mathbf{y}$.

Generalizations of this form have been known since shortly after Shor presented his factoring and discrete logarithm algorithms. For example, in [18] the hidden subgroup problem was discussed for a large class of finite Abelian groups, and more generally in [2] for any finite Abelian group presented as a product of finite cyclic groups. In [13] the Abelian hidden subgroup algorithm was related to eigenvalue estimation.

Other problems which can be formulated in this way include the following.

Deutsch's Problem

Input: A black box that implements $U_f : |x\rangle|b\rangle \mapsto |x\rangle|b \oplus f(x)\rangle$, for some function f that maps $\mathbb{Z}_2 = \{0, 1\}$ to $\{0, 1\}$.
Output: "Constant" if $f(0) = f(1)$, "balanced" if $f(0) \neq f(1)$.

Note that $f(x) = f(y)$ if and only if $x - y \in K$, where K is either $\{0\}$ or $\mathbb{Z}_2 = \{0, 1\}$. If $K = \{0\}$ then f is $1-1$ or "balanced" and if $K = \mathbb{Z}_2$ then f is constant [4,5].

Simon's Problem

Input: A black box that implements $U_f : |\mathbf{x}\rangle|\mathbf{b}\rangle \mapsto |\mathbf{x}\rangle|\mathbf{b} \oplus f(\mathbf{x})\rangle$ for some function f from $\mathbb{Z}_2^n$ to some set X (which is assumed to consist of binary strings of some fixed length) with the property that $f(\mathbf{x}) = f(\mathbf{y})$ if and only if $\mathbf{x} - \mathbf{y} \in K = \{\mathbf{0}, \mathbf{s}\}$ for some $\mathbf{s} \in \mathbb{Z}_2^n$.
Output: The "hidden" string $\mathbf{s}$.

The decision version allows $K = \{\mathbf{0}\}$ and asks whether K is trivial. Simon [16] presented an efficient algorithm for solving this problem, and an exponential lower bound on the query complexity. The solution to the Abelian hidden subgroup problem is a generalization of Simon's algorithm (which deals with finite groups with many generators) and Shor's algorithms [14,12] (which deal with an infinite group with one generator, and a finite group with two generators).

Key Results

Theorem (Abelian stabilizer problem) *There exists a quantum algorithm that, given an instance of the Abelian stabilizer problem, makes $n + O(1)$ queries to $U_{(G,X)}$, uses poly(n) other elementary quantum and classical operations, and with probability at least 2/3 outputs values $h_1, h_2, \ldots, h_l$ such that $St_G(z) = \langle h_1\rangle \oplus \langle h_2\rangle \oplus \cdots \langle h_l\rangle$.*

Kitaev first solved this problem (with a slightly higher query complexity, because his eigenvalue estimation procedure was not optimal). An eigenvalue estimation procedure based on the quantum Fourier transform achieves the $n + O(1)$ query complexity.

Theorem (Abelian hidden subgroup problem) *There exists a quantum algorithm that, given an instance of the Abelian hidden subgroup problem, makes $n + O(1)$ queries to U_f, uses poly(n) other elementary quantum and classical operations, and with probability at least 2/3 outputs values $h_1, h_2, \ldots, h_l$ such that $K = \langle h_1\rangle \oplus \langle h_2\rangle \oplus \cdots \langle h_l\rangle$.*

In some cases, the success probability can be made 1 with the same complexity, and in general the success probability can be made $1 - \epsilon$ using $n + O(\log(1/\epsilon))$ queries and

$poly(n, \log(1/\epsilon))$ other elementary quantum and classical operations.

Applications

Most of these applications in fact were known before the Abelian stabilizer problem or the Abelian hidden subgroup problem were formulated.

Finding the Order of an Element in a Group Let a be an element of a group H (which does *not* need to be Abelian). Consider the function f from $G = \mathbb{Z}$ to the group H where $f(x) = a^x$ for some element a of H. Then $f(x) = f(y)$ if and only if $x - y \in r\mathbb{Z}$. The hidden subgroup is $K = r\mathbb{Z}$ and a generator for K gives the order r of a [14,12].

Discrete Logarithms Let a be an element of a group H (which does *not* need to be Abelian), with $a^r = 1$, and suppose $b = a^k$ from some unknown k. The integer k is called the *discrete logarithm of b to the base a*. Consider the function f from $G = \mathbb{Z}_r \times \mathbb{Z}_r$ to H satisfying $f(x_1, x_2) = a^{x_1} b^{x_2}$. Then $f(x_1, x_2) = f(y_1, y_2)$ if and only if $(x_1, x_2) - (y_1, y_2) \in \{(tk, -t), t = 0, 1, \dots, r-1\}$, which is the subgroup $\langle(k, -1)\rangle$ of $\mathbb{Z}_r \times \mathbb{Z}_r$. Thus, finding a generator for the hidden subgroup K will give the discrete logarithm k. Note that this algorithm works for H equal to the multiplicative group of a finite field, or the additive group of points on an elliptic curve, which are groups that are used in public-key cryptography.

Hidden Linear Functions Let σ be some permutation of $\mathbb{Z}_N$ for some integer N. Let h be a function from $G = \mathbb{Z} \times \mathbb{Z}$ to $\mathbb{Z}_N$, $h(x, y) = x + ay \mod N$. Let $f = \sigma \circ h$. The hidden subgroup of f is $\langle(-a, 1)\rangle$. Boneh and Lipton [1] showed that even if the linear structure of h is hidden (by σ), one can efficiently recover the parameter a with a quantum algorithm.

Self-shift-equivalent Polynomials Given a polynomial P in l variables $X_1, X_2, \dots, X_l$ over $\mathbb{F}_q$, the function f that maps $(a_1, a_2, \dots, a_l) \in \mathbb{F}_q^l$ to $P(X_1 - a_1, X_2 - a_2, \dots, X_l - a_l)$ is constant on cosets of a subgroup K of $\mathbb{F}_q^l$. This subgroup K is the set of shift-self-equivalences of the polynomial P. Grigoriev [8] showed how to compute this subgroup.

Decomposition of a Finitely Generated Group Let G be a group with a unique binary representation for each element of G, and assume that the group operation, and recognizing if a binary string represents an element of G or not, can be done efficiently.

Given a set of generators $g_1, g_2, \dots, g_n$ for a group G, output a set of elements $h_1, h_2, \dots, h_l$, $l \le n$, from the group G such that $G = \langle g_1\rangle \oplus \langle g_2\rangle \oplus \dots \oplus \langle g_l\rangle$. Such a generating set can be found efficiently [3] from generators of the hidden subgroup of the function that maps $(m_1, m_2, \dots, m_n) \mapsto g_1^{m_1} g_2^{m_2} \dots g_n^{m_n}$.

Discussion: What About non-Abelian Groups?

The great success of quantum algorithms for solving the Abelian hidden subgroup problem leads to the natural question of whether it can solve the hidden subgroup problem for non-Abelian groups. It has been shown that a polynomial number of queries suffice [7]; however, in general there is no bound on the overall computational complexity (which includes other elementary quantum or classical operations).

This question has been studied by many researchers, and efficient quantum algorithms can be found for some non-Abelian groups. However, at present, there is no efficient algorithm for most non-Abelian groups. For example, solving the hidden subgroup problem for the symmetric group would directly solve the graph automorphism problem.

Cross References

► Graph Isomorphism
► Quantum Algorithm for the Discrete Logarithm Problem
► Quantum Algorithm for Factoring
► Quantum Algorithm for the Parity Problem
► Quantum Algorithm for Solving the Pell's Equation

Recommended Reading

1. Boneh, D., Lipton, R.: Quantum Cryptanalysis of Hidden Linear Functions (Extended Abstract) In: Proceedings of 15th Annual International Cryptology Conference (CRYPTO'95), pp. 424–437, Santa Barbara, 27–31 August 1995
2. Brassard, G., Høyer, P.: An exact quantum polynomial-time algorithm for Simon's problem. In: Proc. of Fifth Israeli Symposium on Theory of Computing ans Systems (ISTCS'97), pp. 12–23 (1997) and in: Proceedings IEEE Computer Society, Ramat-Gan, 17–19 June 1997
3. Cheung, K., Mosca, M.: Decomposing Finite Abelian Groups. Quantum Inf. Comp. **1**(2), 26–32 (2001)
4. Cleve, R., Ekert, A., Macchiavello, C., Mosca, M.: Quantum Algorithms Revisited. Proc. Royal Soc. London A **454**, 339–354 (1998)
5. Deutsch, D.: Quantum theory, the Church-Turing principle and the universal quantum computer. Proc. Royal Soc. London A **400**, 97–117 (1985)
6. Deutsch, D., Jozsa, R.: Rapid solutions of problems by quantum computation. Proc. Royal Soc. London A **439**, 553–558 (1992)

7. Ettinger, M., Høyer, P., Knill, E.: The quantum query complexity of the hidden subgroup problem is polynomial. Inf. Process. Lett. **91**, 43–48 (2004)
8. Grigoriev, D.: Testing Shift-Equivalence of Polynomials by Deterministic, Probabilistic and Quantum Machines. Theor. Comput. Sci. **180**, 217–228 (1997)
9. Høyer, P.: Conjugated operators in quantum algorithms. Phys. Rev. A **59**(5), 3280–3289 (1999)
10. Kaye, P., Laflamme, R., Mosca, M.: An Introduction to Quantum Computation. Oxford University Press, Oxford (2007)
11. Kitaev, A.: Quantum measurements and the Abelian Stabilizer Problem. quant-ph/9511026, http://arxiv.org/abs/quant-ph/9511026 (1995) and in: Electronic Colloquium on Computational Complexity (ECCC) 3, Report TR96-003,http://eccc.hpi-web.de/eccc-reports/1995/TR96-003/ (1996)
12. Kitaev, A.Y.: Quantum computations: algorithms and error correction. Russ. Math. Surv. **52**(6), 1191–1249 (1997)
13. Mosca, M., Ekert, A.: The Hidden Subgroup Problem and Eigenvalue Estimation on a Quantum Computer. In: Proceedings 1st NASA International Conference on Quantum Computing & Quantum Communications. Lecture Notes in Computer Science, vol. 1509, pp. 174–188. Springer, London (1998)
14. Shor, P.: Algorithms for Quantum Computation: Discrete Logarithms and Factoring. In: Proceedings of the 35th Annual Symposium on Foundations of Computer Science, pp. 124–134, Santa Fe, 20–22 November 1994
15. Shor, P.: Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. SIAM J. Comp. **26**, 1484–1509 (1997)
16. Simon, D.: On the power of quantum computation. In: Proceedings of the 35th IEEE Symposium on the Foundations of Computer Science (FOCS), pp. 116–123, Santa Fe, 20–22 November 1994
17. Simon, D.: On the Power of Quantum Computation. SIAM J. Comp. **26**, 1474–1483 (1997)
18. Vazirani, U.: Berkeley Lecture Notes. Fall 1997. Lecture 8. http://www.cs.berkeley.edu/~vazirani/qc.html (1997)

Adaptive Partitions

1986; Du, Pan, Shing

PING DENG[1], WEILI WU[1], EUGENE SHRAGOWITZ[2]
[1] Department of Computer Science, University of Texas at Dallas, Richardson, TX, USA
[2] Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN, USA

Keywords and Synonyms

Technique for constructing approximation

Problem Definition

Adaptive partition is one of major techniques to design polynomial-time approximation algorithms, especially polynomial-time approximation schemes for geometric optimization problems. The framework of this technique is to put the input data into a rectangle and partition this rectangle into smaller rectangles by a sequence of cuts so that the problem is also partitioned into smaller ones. Associated with each adaptive partition, a feasible solution can be constructed recursively from solutions in smallest rectangles to bigger rectangles. With dynamic programming, an optimal adaptive partition is computed in polynomial time.

Historical Background

The adaptive partition was first introduced to the design of an approximation algorithm by Du et al. [5] with a guillotine cut while they studied the minimum edge length rectangular partition (MELRP) problem. They found that if the partition is performed by a sequence of guillotine cuts, then an optimal solution can be computed in polynomial time with dynamic programming. Moreover, this optimal solution can be used as a pretty good approximation solution for the original rectangular partition problem. Both Arora [1] and Mitchell et al. [12,13] found that the cut needs not to be completely guillotine. In other words, the dynamic programming can still runs in polynomial time if subproblems have some relations but the number of relations is smaller. As the number of relations goes up, the approximation solution obtained approaches the optimal one, while the run time, of course, goes up. They also found that this technique can be applied to many geometric optimization problems to obtain polynomial-time approximation schemes.

Key Results

The MELRP was proposed by Lingas et al. [9] as follows: Given a rectilinear polygon possibly with some rectangular holes, partition it into rectangles with minimum total edge length. Each hole may be degenerated into a line segment or a point.

There are several applications mentioned in [9] for the background of the problem: process control (stock cutting), automatic layout systems for integrated circuit (channel definition), and architecture (internal partitioning into offices). The *minimum edge length* partition is a natural goal for these problems since there is a certain amount of waste (e. g., sawdust) or expense incurred (e. g., for dividing walls in the office) which is proportional to the sum of edge lengths drawn. For very large scale integration (VLSI) design, this criterion is used in the MIT Placement and Interconnect (PI) System to divide the routing region up into channels - one finds that this produces large “natural-looking” channels with a minimum of channel-to-channel interaction to consider.

They showed that while the MELRP in general is nondeterministic polynomial-time (NP) hard, is can be solved in time $O(n^4)$ in the hole-free case, where n is the number of vertices in the input rectilinear polygon. The polynomial algorithm is essentially a dynamic programming based on the fact that there always exists an optimal solution satisfying the property that every cut line passes through a vertex of the input polygon or holes (namely, every maximal cut segment is incident to a vertex of input or holes).

A naive idea to design an approximation algorithm for the general case is to use a forest connecting all holes to the boundary and then to solve the resulting hole-free case in $O(n^4)$ time. With this idea, Lingas [10] gave the first constant-bounded approximation; its performance ratio is 41.

Motivated by a work of Du et al. [4] on application of dynamic programming to optimal routing trees, Du et al. [5] initiated an idea of adaptive partition. They used a sequence of guillotine cuts to do rectangular partition; each guillotine cut breaks a connected area into at least two parts. With dynamic programming, they were able to show that a minimum-length guillotine rectangular partition (i. e., one with minimum total length among all guillotine partitions) can be computed in $O(n^5)$ time. Therefore, they suggested using the minimum-length guillotine rectangular partition to approximate the MELRP and tried to analyze the performance ratio. Unfortunately, they failed to get a constant ratio in general and only obtained a upper bound of 2 for the performance ratio in a NP-hard special case [7]. In this special case, the input is a rectangle with some points inside. Those points are holes. The following is a simple version of the proof obtained by Du et al. [6].

Theorem *The minimum-length guillotine rectangular partition is an approximation with performance ratio 2 for the MELRP.*

Proof Consider a rectangular partition P. Let $proj_x(P)$ denote the total length of segments on a horizontal line covered by vertical projection of the partition P.

A rectangular partition is said to be covered by a guillotine partition if each segment in the rectangular partition is covered by a guillotine cut of the latter. Let $guil(P)$ denote the minimum length of the guillotine partition covering P and $length(P)$ denote the total length of rectangular partition P. It will be proved by induction on the number k of segments in P that

$$guil(P) \leq 2 \cdot length(P) - proj_x(P) \, .$$

For $k = 1$, one has $guil(P) = length(P)$. If the segment is horizontal, then one has $proj_x(P) = length(P)$ and hence

$$guil(P) = 2 \cdot length(P) - proj_x(P) \, .$$

If the segment is vertical, then $proj_x(P) = 0$ and hence

$$guil(P) < 2 \cdot length(P) - proj_x(P) \, .$$

Now, consider $k \geq 2$. Suppose that the initial rectangle has each vertical edge of length a and each horizontal edge of length b. Consider two cases:

Case 1. There exists a vertical segment s having length greater than or equal to $0.5a$. Apply a guillotine cut along this segment s. Then the remainder of P is divided into two parts P_1 and P_2 which form rectangular partition of two resulting small rectangles, respectively. By induction hypothesis,

$$guil(P_i) \leq 2 \cdot length(P_i) - proj_x(P_i)$$

for $i = 1, 2$. Note that

$$\begin{aligned} guil(P) &\leq guil(P_1) + guil(P_2) + a \, , \\ length(P) &= length(P_1) + length(P_2) + length(s) \, , \\ proj_x(P) &= proj_x(P_1) + proj_x(P_2) \, . \end{aligned}$$

Therefore,

$$guil(P) \leq 2 \cdot length(P) - proj_x(P) \, .$$

Case 2. No vertical segment in P has length greater than or equal to $0.5a$. Choose a horizontal guillotine cut which partitions the rectangle into two equal parts. Let P_1 and P_2 denote rectangle partitions of the two parts, obtained from P. By induction hypothesis,

$$guil(P_i) \leq 2 \cdot length(P_i) - proj_x(P_i)$$

for $i = 1, 2$. Note that

$$\begin{aligned} guil(P) &= guil(P_1) + guil(P_2) + b \, , \\ length(P) &\geq length(P_1) + length(P_2) \, , \\ proj_x(P) &= proj_x(P_1) = proj_x(P_2) = b \, . \end{aligned}$$

Therefore,

$$guil(P) \leq 2 \cdot length(P) - proj_x(P) \, .$$

Gonzalez and Zheng [8] improved this upper bound to 1.75 and conjectured that the performance ratio in this case is 1.5.

Applications

In 1996, Arora [1] and Mitchell et al. [12,13,14] found that the cut does not necessarily have to be completely guillotine in order to have a polynomial-time computable optimal solution for such a sequence of cuts. Of course, the number of connections left by an incomplete guillotine cut should be limited. While Mitchell et al. developed the *m*-guillotine subdivision technique, Arora employed a "portal" technique. They also found that their techniques can be used for not only the MELRP, but also for many geometric optimization problems [1,2,3,12,13,14,15].

Open Problems

One current important submicron step of technology evolution in electronics interconnects has become the dominating factor in determining VLSI performance and reliability. Historically a problem of interconnects design in VLSI has been very tightly intertwined with the classical problem in computational geometry: Steiner minimum tree generation. Some essential characteristics of VLSI are roughly proportional to the length of the interconnects. Such characteristics include chip area, yield, power consumption, reliability and timing. For example, the area occupied by interconnects is proportional to their combined length and directly impacts the chip size. Larger chip size results in reduction of yield and increase in manufacturing cost. The costs of other components required for manufacturing also increase with increase of the wire length. From the performance angle, longer interconnects cause an increase in power dissipation, degradation of timing and other undesirable consequences. That is why finding the minimum length of interconnects consistent with other goals and constraints is such an important problem at this stage of VLSI technology.

The combined length of the interconnects on a chip is the sum of the lengths of individual signal nets. Each signal net is a set of electrically connected terminals, where one terminal acts as a driver and other terminals are receivers of electrical signals. Historically, for the purpose of finding an optimal configuration of interconnects, terminals were considered as points on the plane, and a routing problem for individual nets was formulated as a classical Steiner minimum tree problem. For a variety of reasons VLSI technology implements only rectilinear wiring on the set of parallel planes, and, consequently, with few exceptions, only a rectilinear version of the Steiner tree is being considered in the VLSI domain. This problem is known as the RSMT.

Further progress in VLSI technology resulted in more factors than just length of interconnects gaining importance in selection of routing topologies. For example, the presence of obstacles led to reexamination of techniques used in studies of the rectilinear Steiner tree, since many classical techniques do not work in this new environment. To clarify the statement made above, we will consider the construction of a rectilinear Steiner minimum tree in the presence of obstacles.

Let us start with a rectilinear plane with obstacles defined as rectilinear polygons. Given *n* points on the plane, the objective is to find the shortest rectilinear Steiner tree that interconnects them. One already knows that a polynomial-time approximation scheme for RSMT without obstacles exists and can be constructed by adaptive partition with application of either the portal or the *m*-guillotine subdivision technique. However, both the *m*-guillotine cut and the portal techniques do not work in the case that obstacles exists. The portal technique is not applicable because obstacles may block movement of the line that crosses the cut at a portal. The *m*-guillotine cut could not be constructed either, because obstacles may break down the cut segment that makes the Steiner tree connected.

In spite of the facts stated above, the RSMT with obstacles may still have polynomial-time approximation schemes.Strong evidence was given by Min et al. [11]. They constructed a polynomial-time approximation scheme for the problem with obstacles under the condition that the ratio of the longest edge and the shortest edge of the minimum spanning tree is bounded by a constant. This design is based on the classical nonadaptive partition approach. All of the above make us believe that a new adaptive technique can be found for the case with obstacles.

Cross References

► Metric TSP
► Rectilinear Steiner Tree
► Steiner Trees

Recommended Reading

1. Arora, S.: Polynomial-time approximation schemes for Euclidean TSP and other geometric problems. In: Proc. 37th IEEE Symp. on Foundations of Computer Science, 1996, pp. 2–12
2. Arora, S.: Nearly linear time approximation schemes for Euclidean TSP and other geometric problems. In: Proc. 38th IEEE Symp. on Foundations of Computer Science, 1997, pp. 554–563
3. Arora, S.: Polynomial-time approximation schemes for Euclidean TSP and other geometric problems. J. ACM **45**, 753–782 (1998)
4. Du, D.Z., Hwang, F.K., Shing, M.T., Witbold, T.: Optimal routing trees. IEEE Trans. Circuits **35**, 1335–1337 (1988)
5. Du, D.-Z., Pan, L.-Q., Shing, M.-T.: Minimum edge length guillotine rectangular partition. Technical Report 0241886, Math. Sci. Res. Inst., Univ. California, Berkeley (1986)
6. Du, D.-Z., Hsu, D.F., Xu, K.-J.: Bounds on guillotine ratio. Congressus Numerantium **58**, 313–318 (1987)

7. Gonzalez, T., Zheng, S.Q.: Bounds for partitioning rectilinear polygons. In: Proc. 1st Symp. on Computational Geometry (1985)
8. Gonzalez, T., Zheng, S.Q.: Improved bounds for rectangular and guillotine partitions. J. Symb. Comput. **7**, 591–610 (1989)
9. Lingas, A., Pinter, R.Y., Rivest, R.L., Shamir, A.: Minimum edge length partitioning of rectilinear polygons. In: Proc. 20th Allerton Conf. on Comm. Control and Compt., Illinos (1982)
10. Lingas, A.: Heuristics for minimum edge length rectangular partitions of rectilinear figures. In: Proc. 6th GI-Conference, Dortmund, January 1983. Springer
11. Min, M., Huang, S.C.-H., Liu, J., Shragowitz, E., Wu, W., Zhao, Y., Zhao, Y.: An Approximation Scheme for the Rectilinear Steiner Minimum Tree in Presence of Obstructions. Fields Inst. Commun. **37**, 155–164 (2003)
12. Mitchell, J.S.B.: Guillotine subdivisions approximate polygonal subdivisions: A simple new method for the geometric *k*-MST problem. In: Proc. 7th ACM-SIAM Symposium on Discrete Algorithms, 1996, pp. 402–408.
13. Mitchell, J.S.B., Blum, A., Chalasani, P., Vempala, S.: A constant-factor approximation algorithm for the geometric *k*-MST problem in the plane. SIAM J. Comput. **28**(3), 771–781 (1999)
14. Mitchell, J.S.B.: Guillotine subdivisions approximate polygonal subdivisions: Part II – A simple polynomial-time approximation scheme for geometric *k*-MST, TSP, and related problem. SIAM J. Comput. **29**(2), 515–544 (1999)
15. Mitchell, J.S.B.: Guillotine subdivisions approximate polygonal subdivisions: Part III – Faster polynomial-time approximation scheme for geometric network optimization, manuscript, State University of New York, Stony Brook (1997)

Ad-Hoc Networks

► Channel Assignment and Routing in Multi-Radio Wireless Mesh Networks

Adword Auction

► Position Auction

Adwords Pricing

2007; Bu, Deng, Qi

TIAN-MING BU
Department of Computer Science & Engineering,
Fudan University, Shanghai, China

Problem Definition

The model studied here is the same as that which was first presented in [11] by Varian. For some keyword, $\mathcal{N} = \{1, 2, \ldots, N\}$, advertisers bid $\mathcal{K} = \{1, 2, \ldots, K\}$ advertisement slots ($K < N$) which will be displayed on the search result page from top to bottom. The higher the advertisement is positioned, the more conspicuous it is and the more clicks it receives. Thus for any two slots $k_1, k_2 \in \mathcal{K}$, if $k_1 < k_2$, then slot k_1's click-through rate (CTR) c_{k_1} is larger than c_{k_2}. That is, $c_1 > c_2 > \ldots > c_K$, from top to bottom, respectively. Moreover, each bidder $i \in \mathcal{N}$ has privately known information, v^i, which represents the expected return per click to bidder i.

According to each bidder i's submitted bid b^i, the auctioneer then decides how to distribute the advertisement slots among the bidders and how much they should pay per click. In particular, the auctioneer first sorts the bidders in decreasing order according to their submitted bids. Then the highest slot is allocated to the first bidder, the second highest slot is allocated to the second bidder, and so on. The last $N - K$ bidders would lose and get nothing. Finally, each winner would be charged on a per-click basis for the next bid in the descending bid queue. The losers would pay nothing.

Let b_k denote the kth highest bid in the descending bid queue and v_k the true value of the kth bidder in the descending queue. Thus if bidder i got slot k, i's payment would be $b_{k+1} \cdot c_k$. Otherwise, his payment would be zero. Hence, for any bidder $i \in \mathcal{N}$, if i were on slot $k \in \mathcal{K}$, his utility (payoff) could be represented as

$$u^i_k = (v^i - b_{k+1}) \cdot c_k \,.$$

Unlike one-round sealed-bid auctions where each bidder has only one chance to bid, the adword auction allows bidders to change their bids any time. Once bids are changed, the system refreshes the ranking automatically and instantaneously. Accordingly, all bidders' payment and utility are also recalculated. As a result, other bidders could then have an incentive to change their bids to increase their utility, and so on.

Definition 1 (Adword Pricing)
INPUT: the CTR for each slot, each bidder's expected return per click on his advertising.
OUTPUT: the stable states of this auction and whether any of these stable states can be reached from any initial states.

Key Results

Let $\mathbf{b}$ represent the bid vector $(b^1, b^2, \ldots, b^N)$. $\forall i \in \mathcal{N}$, $\mathcal{O}^i(\mathbf{b})$ denotes bidder i's place in the descending bid queue. Let $\mathbf{b}^{-i} = (b^1, \ldots, b^{i-1}, b^{i+1}, \ldots, b^N)$ denote the bids of all other bidders except i. $\mathcal{M}^i(\mathbf{b}^{-i})$ returns a set defined as

$$\mathcal{M}^i(\mathbf{b}^{-i}) = \arg \max_{b^i \in [0, v^i]} \left\{ u^i_{\mathcal{O}^i(b^i, \mathbf{b}^{-i})} \right\} . \qquad (1)$$

Definition 2 (Forward-Looking Best-Response Function) Given $\mathbf{b}^{-i}$, suppose $\mathcal{O}^i(\mathcal{M}^i(\mathbf{b}^{-i}), \mathbf{b}^{-i}) = k$, then bidder i's forward-looking response function $\mathcal{F}^i(\mathbf{b}^{-i})$ is defined as

$$\mathcal{F}^i(\mathbf{b}^{-i}) = \begin{cases} v^i - \frac{c_k}{c_{k-1}}(v^i - b_{k+1}) & 2 \le k \le K\,, \\ v^i & k = 1 \text{ or } k > K\,. \end{cases} \quad (2)$$

Definition 3 (Forward-Looking Nash Equilibrium) A forward-looking best-response-function-based Nash equilibrium is a strategy profile $\hat{\mathbf{b}}$ such that

$$\forall i \in \mathcal{N}\,, \quad \hat{\mathbf{b}}^i \in \mathcal{F}^i(\hat{\mathbf{b}}^{-i})\,.$$

Definition 4 (Output Truthful [7,9]) For any instance of an adword auction and the corresponding equilibrium set $\mathcal{E}$, if $\forall \mathbf{e} \in \mathcal{E}$ and $\forall i \in \mathcal{N}$, $\mathcal{O}^i(\mathbf{e}) = \mathcal{O}^i(v^1, \dots, v^N)$, then the adword auction is *output truthful* on $\mathcal{E}$.

Theorem 5 *An adword auction is output truthful on* $\mathcal{E}_{\text{forward-looking}}$.

Corollary 6 *An adword auction has a unique forward-looking Nash equilibrium.*

Corollary 7 *Any bidder's payment under the forward-looking Nash equilibrium is equal to her payment under the VCG mechanism for the auction.*

Corollary 8 *For adword auctions, the auctioneer's revenue in a forward-looking Nash equilibrium is equal to her revenue under the VCG mechanism for the auction.*

Definition 9 (Simultaneous Readjustment Scheme) In a simultaneous readjustment scheme, all bidders participating in the auction will use forward-looking best-response function $\mathcal{F}$ to update their current bids simultaneously, which turns the current stage into a new stage. Then, based on the new stage, all bidders may update their bids again.

Theorem 10 *An adword auction may not always converge to a forward-looking Nash equilibrium under the simultaneous readjustment scheme even when the number of slots is 3. But the protocol converges when the number of slots is 2.*

Definition 11 (Round-Robin Readjustment Scheme) In the round-robin readjustment scheme, bidders update their biddings one after the other, according to the order of the bidder's number or the order of the slots.

Theorem 12 *An adword auction may not always converge to a forward-looking Nash equilibrium under the round-robin readjustment scheme even when the number of slots is 4. But the protocol converges when the number of slots is 2 or 3.*

```
1:  if ($j = 0$) then
2:      exit
3:  end if
4:  Let $i$ be the ID of the bidder whose current bid is $b_j$
        (and equivalently, $b^i$).
5:  Let $h = \mathcal{O}^i(\mathcal{M}^i(\mathbf{b}^{-i}), \mathbf{b}^{-i})$.
6:  Let $\mathcal{F}^i(\mathbf{b}^{-i})$ be the best response function value for
        Bidder $i$.
7:  Re-sort the bid sequence. (So $h$ is the slot of the new
        bid $\mathcal{F}^i(\mathbf{b}^{-i})$ of Bidder $i$.)
8:  if ($h < j$) then
9:      call Lowest-First($K, j, b_1, b_2, \cdots, b_N$),
10: else
11:     call Lowest-First($K, h-1, b_1, b_2, \cdots, b_N$)
12: end if
```

Adwords Pricing, Figure 1
Readjustment Scheme: Lowest-First($K, j, b_1, b_2, \cdots, b_N$)

Theorem 13 *Adword auctions converge to a forward-looking Nash equilibrium in finite steps with a* lowest-first *adjustment scheme.*

Theorem 14 *Adword auctions converge to a forward-looking Nash equilibrium with probability one under a randomized readjustment scheme.*

Applications

Online adword auctions are the fastest growing form of advertising on the Internet today. Many search engine companies such as Google and Yahoo! make huge profits on this kind of auction. Because advertisers can change their bids any time, such auctions can reduce advertisers' risk. Further, because the advertisement is only displayed to those people who are really interested in it, such auctions can reduce advertisers' investment and increase their return on investment.

For the same model, Varian [11] focuses on a subset of Nash equilibrium called *symmetric Nash equilibrium*, which can be formulated nicely and dealt with easily. Edelman et al. [8] study *locally envy-free* equilibrium, where no player can improve her payoff by exchanging bid with the player ranked one position above her. Coincidently, locally envy-free equilibrium is equal to symmetric Nash equilibrium proposed in [11]. Further, the revenue under the forward-looking Nash equilibrium is the same as the lower bound under Varian's symmetric Nash equilibrium and the lower bound under Edelman et al.'s locally envy-free equilibrium. In [6], Cary et al. also study the dynamic model's equilibrium and convergence based on the *balanced bidding strategy*, which is actually the same as the *forward-looking best-response function* in [4]. Cary et al. explore the convergence properties under two models, a *synchronous* model, which is the same as the *simultaneous readjustment scheme* in [4], and an *asynchronous* model, which is the same as the *randomized readjustment scheme* in [4].

In addition, there are other models for adword auctions. [1] and [5] study the model under which each bidder can submit a daily budget, even the maximum number of clicks per day, in addition to the price per click. Both [10] and [3] study bidders' behavior of bidding on several keywords. [2] studies a model whereby the advertiser not only submits a bid but additionally submits which positions he is going to bid for.

Open Problems

The speed of convergence remains open. Does the dynamic model converge in polynomial time under randomized readjustment scheme? Even more, are there other readjustment schemes that converge in polynomial time?

Cross References

► Multiple Unit Auctions with Budget Constraint
► Position Auction

Recommended Reading

1. Abrams, Z.: Revenue maximization when bidders have budgets. In: Proceedings of the 17th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA-06), Miami, FL 2006, pp. 1074–1082, ACM Press, New York (2006)
2. Aggarwal, G., Muthukrishnan, S., Feldman, J.: Bidding to the top: Vcg and equilibria of position-based auctions. http://www.citebase.org/abstract?id=oai:arXiv.org:cs/0607117 (2006)
3. Borgs, C., Chayes, J., Etesami, O., Immorlica, N., Jain, K., Mahdian, M.: Bid optimization in online advertisement auctions. In: 2nd Workshop on Sponsored Search Auctions, in conjunction with the ACM Conference on Electronic Commerce (EC-06), Ann Arbor, MI, 2006
4. Bu, T.-M., Deng, X., Qi, Q.: Dynamics of strategic manipulation in ad-words auction. In: 3rd Workshop on Sponsored Search Auctions, in conjunction with WWW2007, Banff, Canada, 2007
5. Bu, T.-M., Qi, Q., Sun, A.W.: Unconditional competitive auctions with copy and budget constraints. In: Spirakis, P.G., Mavronicolas, M., Kontogiannis, S.C. (eds.) Internet and Network Economics, 2nd International Workshop, WINE 2006. Lecture Notes in Computer Science, vol. 4286, pp. 16–26, Patras, Greece, December 15–17. Springer, Berlin (2006)
6. Cary, M., Das, A., Edelman, B., Giotis, I., Heimerl, K., Karlin, A.R., Mathieu, C., Schwarz, M.: Greedy bidding strategies for keyword auctions. In: MacKie-Mason, J.K., Parkes, D.C., Resnick, P. (eds.) Proceedings of the 8th ACM Conference on Electronic Commerce (EC-2007), San Diego, California, USA, June 11–15 2007, pp. 262–271. ACM, New York (2007)
7. Chen, X., Deng, X., Liu, B.J.: On incentive compatible competitive selection protocol. In: Computing and Combinatorics, 12th Annual International Conference, COCOON 2006, Taipei, Taiwan, 15 August 2006. Lecture Notes in Computer Science, vol. 4112, pp. 13–22. Springer, Berlin (2006)
8. Edelman, B., Ostrovsky, M., Schwarz, M.: Internet advertising and the generalized second price auction: selling billions of dollars worth of dollars worth of keywords. In: 2nd Workshop on Sponsored Search Auctions, in conjunction with the ACM Conference on Electronic Commerce (EC-06), Ann Arbor, MI, June 2006
9. Kao, M.-Y., Li, X.-Y., Wang, W.: Output truthful versus input truthful: a new concept for algorithmic mechanism design (2006)
10. Kitts, B., Leblanc, B.: Optimal bidding on keyword auctions. Electronic Markets, Special issue: Innovative Auction Markets **14**(3), 186–201 (2004)
11. Varian, H.R.: Position auctions. Int. J. Ind. Organ. **25**(6), 1163–1178 (2007) http://www.sims.berkeley.edu/~hal/Papers/2006/position.pdf. Accessed 29 March 2006

Agreement

► Asynchronous Consensus Impossibility
► Consensus with Partial Synchrony
► Randomization in Distributed Computing

Algorithm DC-Tree for *k* Servers on Trees

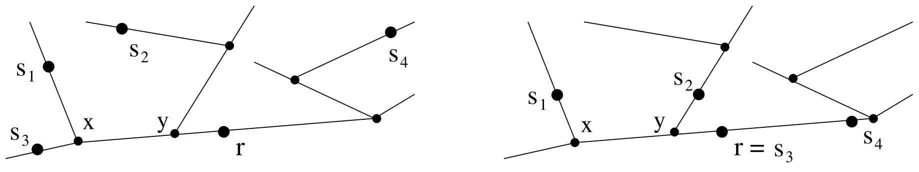
1991; Chrobak, Larmore

MAREK CHROBAK
Department of Computer Science,
University of California, Riverside, CA, USA

Problem Definition

In the *k-server problem*, one wishes to schedule the movement of k servers in a metric space $\mathbb{M}$, in response to a sequence $\varrho = r_1, r_2, \dots, r_n$ of *requests*, where $r_i \in \mathbb{M}$ for each i. Initially, all the servers are located at some point $r_0 \in \mathbb{M}$. After each request r_i is issued, one of the k servers must move to r_i. A *schedule* specifies which server moves to each request. The *cost* of a schedule is the total distance traveled by the servers, and our objective is to find a schedule with minimum cost.

In the *online* version of the k-server problem the decision as to which server to move to each request r_i must be made before the next request r_{i+1} is issued. In other words, the choice of this server is a function of requests



Algorithm DC-Tree for *k* Servers on Trees, Figure 1
Algorithm DC-TREE serving a request on *r*. The initial configuration is on the *left*; the configuration after the service is completed is on the *right*. At first, all servers are active. When server 3 reaches point *x*, server 1 becomes inactive. When server 3 reaches point *y*, server 2 becomes inactive

$r_1, r_2, \ldots, r_i$. It is quite easy to see that in this online scenario it is not possible to guarantee an optimal schedule. The accuracy of online algorithms is often measured using competitive analysis. If $\mathcal{A}$ is an online k-server algorithm, denote by $cost_{\mathcal{A}}(\varrho)$ the cost of the schedule produced by $\mathcal{A}$ on a request sequence ϱ, and by $opt(\varrho)$ the cost of the optimal schedule. $\mathcal{A}$ is called *R-competitive* if $cost_{\mathcal{A}}(\varrho) \leq R \cdot opt(\varrho) + B$, where B is a constant that may depend on $\mathbb{M}$ and r_0. The smallest such R is called the *competitive ratio* of $\mathcal{A}$. Of course, the smaller the R the better.

The k-server problem was introduced by Manasse, McGeoch, and Sleator [7,8], who proved that there is no online R-competitive algorithm for $R < k$, for any metric space with at least $k+1$ points. They also gave a 2-competitive algorithm for $k = 2$ and formulated what is now known as the *k-server conjecture*, which postulates that there exists a k-competitive online algorithm for all k. Koutsoupias and Papadimitriou [5,6] proved that the so-called *work-function algorithm* has competitive ratio at most $2k - 1$, which to date remains the best upper bound known.

Efforts to prove the k-server conjecture led to discoveries of k-competitive algorithms for some restricted classes of metric spaces, including Algorithm DC-TREE for trees [4] presented in the next section. (See [1,2,3] for other examples.) A *tree* is a metric space defined by a connected acyclic graph whose edges are treated as line segments of arbitrary positive lengths. This metric space includes both the tree's vertices and the points on the edges, and the distances are measured along the (unique) shortest paths.

Key Results

Let $\mathbb{T}$ be a tree, as defined above. Given the current server configuration $S = \{s_1, \ldots, s_k\}$, where s_j denotes the location of server j, and a request point r, the algorithm will move several servers, with one of them ending up on r. For two points $x, y \in \mathbb{T}$, let $[x, y]$ be the unique path from x to y in $\mathbb{T}$. A server j is called *active* if there is no other server in $[s_j, r] - \{s_j\}$, and j is the minimum-index server located on s_j (the last condition is needed only to break ties).

Algorithm DC-TREE

On a request r, move all active servers, continuously and with the same speed, towards r, until one of them reaches the request. Note that during this process some active servers may become inactive, in which case they halt. Clearly, the server that will arrive at r is the one that was closest to r at the time when r was issued. Figure 1 shows how DC-TREE serves a request r.

The competitive analysis of Algorithm DC-TREE is based on a potential argument. The cost of Algorithm DC-TREE is compared to that of an adversary who serves the requests with her own servers. Denoting by A the configuration of the adversary servers at a given step, define the potential by $\Phi = k \cdot D(S, A) + \sum_{i<j} d(s_i, s_j)$, where $D(S, A)$ is the cost of the minimum matching between S and A. At each step, the adversary first moves one of her servers to r. In this sub-step the potential increases by at most k times the increase of the adversary's cost. Then, Algorithm DC-TREE serves the request. One can show that then the sum of Φ and DC-TREE's cost does not increase. These two facts, by amortization over the whole request sequence, imply the following result [4]:

Theorem ([4]) *Algorithm* DC-TREE *is k-competitive on trees.*

Applications

The k-server problem is an abstraction of various scheduling problems, including emergency crew scheduling, caching in multilevel memory systems, or scheduling head movement in 2-headed disks. Nevertheless, due to its abstract nature, the k-server problem is mainly of theoretical interest.

Algorithm DC-TREE can be applied to other spaces by "embedding" them into trees. For example, a uniform metric space (with all distances equal 1) can be represented by a star with arms of length 1/2, and thus Algorithm DC-TREE can be applied to those spaces. This also immediately gives a k-competitive algorithm for the *caching problem*, where the objective is to manage a two-level memory system consisting of a large main memory and a cache that can store up to k memory items. If an item is in the cache, it can be accessed at cost 0, otherwise it costs 1 to read it from the main memory. This caching problem can be thought of as the k-server problem in a uniform metric space where the server positions represent the items residing in the cache. This idea can be extended further to the *weighted caching* [3], which is a generalization of the caching problem where different items may have different costs. In fact, if one can embed a metric space $\mathbb{M}$ into a tree with distortion bounded by δ, then Algorithm DC-TREE yields a δk-competitive algorithm for $\mathbb{M}$.

Open Problems

The k-server conjecture – whether there is a k-competitive algorithm for k servers in any metric space – remains open. It would be of interest to prove it for some natural special cases, for example the plane, either with the Euclidean or Manhattan metric. (A k-competitive algorithm for the Manhattan plane for $k = 2, 3$ servers is known [1], but not for $k \geq 4$.)

Very little is known about online *randomized* algorithms for k-servers. In fact, even for $k = 2$ it is not known if there is a randomized algorithm with competitive ratio smaller than 2.

Cross References

▶ Deterministic Searching on the Line
▶ Generalized Two-Server Problem
▶ Metrical Task Systems
▶ Online Paging and Caching
▶ Paging
▶ Work-Function Algorithm for k Servers

Recommended Reading

1. Bein, W., Chrobak, M., Larmore, L.L.: The 3-server problem in the plane. Theor. Comput. Sci. **287**, 387–391 (2002)
2. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press, Cambridge (1998)
3. Chrobak, M., Karloff, H., Payne, T.H., Vishwanathan, S.: New results on server problems. SIAM J. Discret. Math. **4**, 172–181 (1991)
4. Chrobak, M., Larmore, L.L.: An optimal online algorithm for k servers on trees. SIAM J. Comput. **20**, 144–148 (1991)
5. Koutsoupias, E., Papadimitriou, C.: On the k-server conjecture. In: Proc. 26th Symp. Theory of Computing (STOC), pp. 507–511. ACM (1994)
6. Koutsoupias, E., Papadimitriou, C.: On the k-server conjecture. J. ACM **42**, 971–983 (1995)
7. Manasse, M., McGeoch, L.A., Sleator, D.: Competitive algorithms for online problems. In: Proc. 20th Symp. Theory of Computing (STOC), pp. 322–333. ACM (1988)
8. Manasse, M., McGeoch, L.A., Sleator, D.: Competitive algorithms for server problems. J. Algorithms **11**, 208–230 (1990)

Algorithmic Cooling

1999; Schulman, Vazirani
2002; Boykin, Mor, Roychowdhury, Vatan, Vrijen

TAL MOR
Department of Computer Science, Technion, Haifa, Israel

Keywords and Synonyms

Algorithmic cooling of spins; Heat-bath algorithmic cooling

Problem Definition

The fusion of concepts taken from the fields of quantum computation, data compression, and thermodynamics, has recently yielded novel algorithms that resolve problems in nuclear magnetic resonance and potentially in other areas as well; algorithms that "cool down" physical systems.

- A leading candidate technology for the construction of quantum computers is Nuclear Magnetic Resonance (NMR). This technology has the advantage of being well-established for other purposes, such as chemistry and medicine. Hence, it does not require new and exotic equipment, in contrast to ion traps and optical lattices, to name a few. However, when using standard NMR techniques (not only for quantum computing purposes) one has to live with the fact that the state can only be initialized in a very noisy manner: The particles' spins point in mostly random directions, with only a tiny bias towards the desired state.

  The key idea of Schulman and Vazirani [13] is to combine the tools of both data compression and quantum computation, to suggest a *scalable* state initialization process, a "molecular-scale heat engine". Based on Schulman and Vazirani's method, Boykin, Mor, Roychowdhury, Vatan, and Vrijen [2] then developed a new process, "heat-bath algorithmic cooling", to significantly improve the state initialization process, by opening the system to the environment. Strikingly, this offered a way to put to good use the phenomenon of decoherence, which is usually considered to be the villain in quantum computation. These two methods are now sometimes called "closed-system" (or "reversible") algorithmic cooling, and "open-system" algorithmic cooling, respectively.

- The far-reaching consequence of this research lies in the possibility of reaching beyond the potential implementation of remote-future quantum computing devices. An efficient technique to generate ensembles of spins that are highly polarized by external magnetic fields is considered to be a Holy Grail in NMR spectroscopy. Spin-half nuclei have steady-state polarization biases that increase inversely with temperature; therefore, spins exhibiting polarization biases above their thermal-equilibrium biases are considered *cool*. Such cooled spins present an improved signal-to-noise ratio if used in NMR spectroscopy or imaging.
  Existing spin-cooling techniques are limited in their efficiency and usefulness. Algorithmic cooling is a promising new spin-cooling approach that employs data compression methods in *open systems*. It reduces the entropy of spins to a point far beyond Shannon's entropy bound on reversible entropy manipulations, thus increasing their polarization biases. As a result, it is conceivable that the open-system algorithmic cooling technique could be harnessed to improve on *current uses of NMR* in areas such as chemistry, material science, and even medicine, since NMR is at the basis of MRI – Magnetic Resonance Imaging.

Basic Concepts

Loss-Less in-Place Data Compression Given a bit-string of length n, such that the probability distribution is known and far enough from the uniform distribution, one can use data compression to generate a shorter string, say of m bits, such that the entropy of each bit is much closer to one. As a simple example, consider a four-bit-string which is distributed as follows; $p_{0001} = p_{0010} = p_{0100} = p_{1000} = 1/4$, with p_i the probability of the string i. The probability of any other string value is exactly zero, so the probabilities sum up to one. Then, the bit-string can be compressed, via a loss-less compression algorithm, into a 2-bit string that holds the binary description of the location of "1" in the above four strings. As the probabilities of all these strings are zero, one can also envision a similar process that generates an output which is of the same length n as the input, but such that the entropy is compressed via a loss-less, in-place, data compression into the last two bits. For instance, logical gates that operate on the bits can perform the permutation $0001 \to 0000$, $0010 \to 0001$, $0100 \to 0010$ and $1000 \to 0011$, while the other input strings transform to output strings in which the two most significant bits are not zero; for instance $1100 \to 1010$. One can easily see that the entropy is now fully concentrated on the two least significant bits, which are useful in data compression, while the two most significant bits have zero entropy.

In order to gain some intuition about the design of logical gates that perform entropy manipulations, one can look at a closely related scenario which was first considered by von Neumann. He showed a method to extract fair coin flips, given a biased coin; he suggested taking a pair of biased coin flips, with results a and b, and using the value of a **conditioned on** $a \neq b$. A simple calculation shows that $a = 0$ and $a = 1$ are now obtained with equal probabilities, and therefore the entropy of coin a is increased in this case to 1. The opposite case, the probability distribution of a given that $a = b$, results in a highly determined coin flip; namely, a (conditioned) coin-flip with a higher bias or lower entropy. A gate that flips the value of b if (and only if) $a = 1$ is called a Controlled-NOT gate. If after applying such a gate $b = 1$ is obtained, this means that $a \neq b$ prior to the gate operation, thus now the entropy of a is 1. If, on the other hand, after applying such a gate $b = 0$ is obtained, this means that $a = b$ prior to the gate operation, thus the entropy of a is now lower than its initial value.

Spin Temperature, Polarization Bias, and Effective Cooling In physics, two-level systems, namely systems that possess only binary values, are useful in many ways. Often it is important to initialize such systems to a pure state '0' or to a probability distribution which is as close as possible to a pure state '0'. In these physical two-level systems a data compression process that brings some of them closer to a pure state can be considered as "cooling". For quantum two-level systems there is a simple connection between temperature, entropy, and population probability. The population-probability difference between these two levels is known as the polarization bias, ϵ. Consider a single spin-half particle – for instance a hydrogen nucleus – in a constant magnetic field. At equilibrium with a thermal heat-bath the probability of this spin to be up or down (i. e., parallel or anti-parallel to the field direction) is given by: $p_\uparrow = \frac{1+\epsilon}{2}$, and $p_\downarrow = \frac{1-\epsilon}{2}$. The entropy H of the spin is $H(\text{single-bit}) = H(1/2 + \epsilon/2)$ with $H(P) \equiv -P \log_2 P - (1-P) \log_2 (1-P)$ measured in bits. The two pure states of a spin-half nucleus are commonly written as $|\uparrow\rangle \equiv$'0' and $|\downarrow\rangle \equiv$'1'; the $|\rangle$ notation will be clarified elsewhere[1]. The polarization bias of the spin at thermal equilibrium is given by $\epsilon = p_\uparrow - p_\downarrow$. For such a physical system the bias is obtained via a quantum statistical mechanics argument, $\epsilon = \tanh\left(\frac{\hbar \gamma B}{2K_B T}\right)$, where $\hbar$ is Planck's constant, B is the magnetic field, γ is the

[1] Quantum Computing entries in this encyclopedia, e.g. ► Quantum Dense Coding

particle-dependent gyromagnetic constant[2], K_B is Boltzman's coefficient, and T is the thermal heat-bath temperature. For high temperatures or small biases $\epsilon \approx \frac{\hbar\gamma B}{2K_B T}$, thus the bias is inversely proportional to the temperature. Typical values of ϵ for spin-half nuclei at room temperature (and magnetic field of $\sim$ 10 Tesla) are 10^{-5}–10^{-6}, and therefore most of the analysis here is done under the assumption that $\epsilon \ll 1$. The spin temperature at equilibrium is thus $T = \frac{\text{Const}}{\epsilon}$, and its (Shannon) entropy is $H = 1 - (\epsilon^2/\ln 4)$.

A spin temperature out of thermal equilibrium is still defined via the same formulas. Therefore, when a system is moved away from thermal equilibrium, achieving a greater polarization bias is equivalent to cooling the spins *without cooling the system*, and to decreasing their entropy. The process of increasing the bias (reducing the entropy) without decreasing the temperature of the thermal-bath is known as "effective cooling". After a typical period of time, termed the thermalization time or relaxation time, the bias will gradually revert to its thermal equilibrium value; yet during this process, typically in the order of seconds, the effectively-cooled spin may be used for various purposes as described in Sect. "Applications".

Consider a molecule that contains n adjacent spin-half nuclei arranged in a line; these form the bits of the string. These spins are initially at thermal equilibrium due to their interaction with the environment. At room temperature the bits at thermal equilibrium are not correlated to their neighbors on the same string: More precisely, the correlation is very small and can be ignored. Furthermore, in a liquid state one can also neglect the interaction between strings (between molecules). It is convenient to write the probability distribution of a single spin at thermal equilibrium using the "density matrix" notation

$$\rho_\epsilon = \begin{pmatrix} p_\uparrow & 0 \\ 0 & p_\downarrow \end{pmatrix} = \begin{pmatrix} (1+\epsilon)/2 & 0 \\ 0 & (1-\epsilon)/2 \end{pmatrix} , \qquad (1)$$

since these two-level systems are of a quantum nature (namely, these are quantum bits – qubits), and in general, can also have states other than just a classical probability distribution over '0' and '1'. The classical case will now be considered, where ρ contains only diagonal elements and these describe a conventional probability distribution. At thermal equilibrium, the state of $n = 2$ uncorrelated qubits that have the same polarization bias is described by the density matrix $\rho_{\text{init}}^{\{n=2\}} = \rho_\epsilon \otimes \rho_\epsilon$, where $\otimes$ means tensor product. The probability of the state '00', for instance, is then $(1+\epsilon)/2 \times (1+\epsilon)/2 = (1+\epsilon)^2/4$ (etc.). Similarly, the initial state of an n-qubit system of this type, at thermal equilibrium, is

$$\rho_{\text{init}}^{\{n\}} = \rho_\epsilon \otimes \rho_\epsilon \otimes \cdots \otimes \rho_\epsilon . \qquad (2)$$

This state represents a thermal probability distribution, such that the probability of the classical state '000...0' is $P_{000\ldots0} = (1+\epsilon_0)^n/2^n$, etc. In reality, the initial bias is not the same on each qubit[3], but as long as the differences between these biases are small (e. g., all qubits are of the same nucleus), these differences can be ignored in a discussion of an idealized scenario.

Key Results

Molecular Scale Heat Engines

Schulman and Vazirani (SV) [13] identified the importance of in-place loss-less data compression and of the low-entropy bits created in that process: Physical two-level systems (e. g., spin-half nuclei) may be similarly cooled by data compression algorithms. SV analyzed the cooling of such a system using various tools of data compression. A loss-less compression of an n-bit binary string distributed according to the thermal equilibrium distribution, Eq. (2), is readily analyzed using information-theoretical tools: In an ideal compression scheme (not necessarily realizable), with sufficiently large n, all randomness – and hence all the entropy – of the bit string is transferred to $n - m$ bits; the remaining m bits are thus left, with extremely high probability, at a known deterministic state, say the string '000...0'. The entropy H of the entire system is $H(\text{system}) = nH(\text{single} - \text{bit}) = nH(1/2 + \epsilon/2)$. Any compression scheme cannot decrease this entropy, hence Shannon's source coding entropy bound yields $m \leq n[1 - H(1/2 + \epsilon/2)]$. A simple leading-order calculation shows that m is bounded by (approximately) $\frac{\epsilon^2}{2\ln 2} n$ for small values of the initial bias ϵ. Therefore, with typical $\epsilon \sim 10^{-5}$, molecules containing an order of magnitude of 10^{10} spins are required to cool a single spin close to zero temperature.

Conventional methods for NMR quantum computing are based on unscalable state-initialization schemes [5,9] (e. g., the "pseudo-pure-state" approach) in which the signal-to-noise ratio falls exponentially with n, the number of spins. Consequently, these methods are deemed inappropriate for future NMR quantum computers. SV [13] were first to employ tools of information theory to address

[2]This constant, γ, is thus responsible for the difference in equilibrium polarization bias [e. g., a hydrogen nucleus is 4 times more polarized than a carbon isotope ^{13}C nucleus, but about 10^3 less polarized than an electron spin].

[3]Furthermore, individual addressing of each spin during the algorithm requires a slightly different bias for each.

the scaling problem; they presented a compression scheme in which the number of cooled spins scales well (namely, a constant times n). SV also demonstrated a scheme approaching Shannon's entropy bound, for very large n. They provided detailed analyses of three cooling algorithms, each useful for a different regime of ϵ values.

Some ideas of SV were already explored a few years earlier by Sørensen [14], a physical chemist who analyzed effective cooling of spins. He considered the entropy of several spin systems and the limits imposed on cooling these systems by polarization transfer and more general polarization manipulations. Furthermore, he considered spin-cooling processes in which only unitary operations were used, wherein unitary matrices are applied to the density matrices; such operations are realizable, at least from a conceptual point of view. Sørensen derived a stricter bound on unitary cooling, which today bears his name. Yet, unlike SV, he did not infer the connection to data compression or advocate compression algorithms.

SV named their concept "molecular-scale heat engine". When combined with conventional polarization transfer (which is partially similar to a SWAP gate between two qubits), the term "reversible polarization compression (RPC)" to be more descriptive.

Heat-Bath Algorithmic Cooling

The next significant development came when Boykin, Mor, Roychowdhury, Vatan and Vrijen, (hereinafter referred to as BMRVV), invented a new spin-cooling technique, which they named *Algorithmic cooling* [2], or more specifically, heat-bath algorithmic cooling in which the use of controlled interactions with a heat bath enhances the cooling techniques much further. Algorithmic Cooling (AC) expands the effective cooling techniques by exploiting entropy manipulations in *open systems*. It combines RPC steps[4] with fast relaxation (namely, thermalization) of the *hotter spins*, as a way of pumping entropy outside the system and cooling the system *much beyond Shannon's entropy bound*. In order to pump entropy out of the system, AC employs regular spins (here called computation spins) together with rapidly relaxing spins. The latter are auxiliary spins that return to their thermal equilibrium state very rapidly. These spins have been termed "reset spins", or, equivalently, reset bits. The controlled interactions with the heat bath are generated by polarization transfer or by standard algorithmic techniques (of data compression) that transfer the entropy onto the reset spins which then lose this excess entropy into the environment.

[4]When the entire process is RPC, namely, any of the processes that follow SV ideas, one can refer to it as reversible AC or closed-system AC, rather than as RPC.

The ratio $R_{\text{relax-times}}$, between the relaxation time of the computation spins and the relaxation time of the reset spins, must satisfy $R_{\text{relax-times}} \gg 1$. This condition is vital if one wishes to perform many cooling steps on the system to obtain significant cooling.

From a pure information-theoretical point of view, it is legitimate to assume that the only restriction on ideal RPC steps is Shannon's entropy bound; then the equivalent of Shannon's entropy bound, when an ideal open-system AC is used, is that all computation spins can be cooled down to zero temperature, that is to $\epsilon = 1$. Proof. – repeat the following till the entropy of all computation spins is exactly zero: (i) push entropy from computation spins into reset spins; (ii) let the reset spins cool back to room temperature. Clearly, each application of step (i), except the last one, pushes the same amount of entropy onto the reset spins, and then this entropy is removed from the system in step (ii). Of course, a realistic scenario must take other parameters into account such as finite relaxation-time ratios, realistic environment, and physical operations on the spins. Once this is done, cooling to zero temperature is no longer attainable. While finite relaxation times and a realistic environment are system dependent, the constraint of using physical operations is conceptual.

BMRVV therefore pursued an algorithm that follows some physical rules, it is performed by unitary operations and reset steps, and still bypass Shannon's entropy bound, by far. The BMRVV cooling algorithm obtains significant cooling beyond that entropy bound by making use of very long molecules bearing hundreds or even thousands of spins, because its analysis relies on the law of large numbers.

Practicable Algorithmic Cooling

The concept of algorithmic cooling then led to practicable algorithms [8] for cooling *small molecules*. In order to see the impact of practicable algorithmic cooling, it is best to use a different variant of the entropy bound. Consider a system containing n spin-half particles with total entropy higher than $n - 1$, so that there is no way to cool even one spin to zero temperature. In this case, the entropy bound is a result of the compression of the entropy into $n - 1$ fully-random spins, so that the remaining entropy on the last spin is minimal. The entropy of the remaining single spin satisfies $H(\text{single}) \geq 1 - n\epsilon^2/\ln 4$, thus, at most, its polarization can be improved to

$$\epsilon_{\text{final}} \leq \epsilon\sqrt{n}\,. \tag{3}$$

The practicable algorithmic cooling (PAC), suggested by Fernandez, Lloyd, Mor, and Roychowdhury in [8], indicated potential for a near-future application to NMR spectroscopy. In particular, it presented an algorithm named PAC2 which uses any (odd) number of spins n, such that one of them is a reset spin, and $(n-1)$ are computation spins. PAC2 cools the spins such that the coldest one can (approximately) reach a bias amplification by a factor of $(3/2)^{(n-1)/2}$. The approximation is valid as long as the final bias $(3/2)^{(n-1)/2}\epsilon$ is much smaller than 1. Otherwise, a more precise treatment must be done. This proves an exponential advantage of AC over the best possible reversible AC, as these reversible cooling techniques, e. g., of [13,14], are limited to improve the bias by no more than a factor of $\sqrt{n}$. PAC can be applied for small n (e. g., in the range of 10–20), and therefore it is potentially suitable for near-future applications [6,8,10] in chemical and biomedical usages of NMR spectroscopy.

It is important to note that in typical scenarios the initial polarization bias of a reset spin is higher than that of a computation spin. In this case, the bias amplification factor of $(3/2)^{(n-1)/2}$ is relative to the larger bias, that of the reset spin.

Exhaustive Algorithmic Cooling

Next, AC was analyzed, wherein the cooling steps (reset and RPC) are repeated an arbitrary number of times. This is actually an idealization where an unbounded number of reset and logic steps can be applied without error or decoherence, while the computation qubits do not lose their polarization biases. Fernandez [7] considered two computation spins and a single reset spin (the least significant bit, namely the qubit at the right in the tensor-product density-matrix notation) and analyzed optimal cooling of this system. By repeating the reset and compression exhaustively, he realized that the bound on the final biases of the three spins is approximately $\{2, 1, 1\}$ in units of ϵ, the polarization bias of the reset spin.

Mor and Weinstein generalized this analysis further and found that $n-1$ computation spins and a single reset spin can be cooled (approximately) to biases according to the Fibonacci series: {... 34, 21, 13, 8, 5, 3, 2, 1, 1}. The computation spin that is furthest from the reset spin can be cooled up to the relevant Fibonacci number F_n. That approximation is valid as long as the largest term times ϵ is still much smaller than 1. Schulman then suggested the "partner pairing algorithm" (PPA) and proved the optimality of the PPA among all *classical and quantum* algorithms. These two algorithms, the Fibonacci AC and the PPA, led to two joint papers [11,12], where upper and lower bounds on AC were also obtained. The PPA is defined as follows; repeat these two steps until cooling sufficiently close to the limit: (a) RESET – applied to a reset spin in a system containing $n-1$ computation spins and a single (the LSB) reset spin. (b) SORT – a permutation that sorts the 2^n diagonal elements of the density matrix by decreasing order, so that the MSB spin becomes the coldest. Two important theorems proven in [12] are: 1. Lower bound: When $\epsilon 2^n \gg 1$ (namely, for long enough molecules), Theorem 3 in [12] promises that $n - \log(1/\epsilon)$ cold qubits can be extracted. This case is relevant for scalable NMR quantum computing. 2. Upper bound: Section 4.2 in [12] proves the following theorem: No algorithmic cooling method can increase the probability of any basis state to above $\min\{2^{-n} e^{2^n \epsilon}, 1\}$, wherein the initial configuration is the completely mixed state (the same is true if the initial state is a thermal state).

More recently, Elias, Fernandez, Mor, and Weinstein [6] analyzed more closely the case of $n < 15$ (at room temperature), where the coldest spin (at all stages) still has a polarization bias much smaller than 1. This case is most relevant for near-future applications in NMR spectroscopy. They generalized the Fibonacci-AC to algorithms yielding higher-term Fibonacci series, such as the tri-bonacci (also known as 3-term Fibonacci series), {... 81, 44, 24, 13, 7, 4, 2, 1, 1}, etc. The ultimate limit of these multi-term Fibonacci series is obtained when each term in the series is the sum of all previous terms. The resulting series is precisely the exponential series {... 128, 64, 32, 16, 8, 4, 2, 1, 1}, so the coldest spin is cooled by a factor of 2^{n-2}. Furthermore, a leading order analysis of the upper bound mentioned above (Section 4.2 in [12]) shows that no spin can be cooled beyond a factor of 2^{n-1}; see Corollary 1 in [6].

Applications

The two major far-future and near-future applications are already described in Sect. "Problem Definition". It is important to add here that although the specific algorithms analyzed so far for AC are usually classical, their practical implementation via an NMR spectrometer must be done through analysis of universal quantum computation, using the specific gates allowed in such systems. Therefore, AC could yield the first near-future application of quantum computing devices.

AC may also be useful for cooling various other physical systems, since state initialization is a common problem in physics in general and in quantum computation in particular.

Open Problems

A main open problem in practical AC is technological; can the ratio of relaxation times be increased so that many cooling steps may be applied onto relevant NMR systems? Other methods, for instance a spin-diffusion mechanism [1], may also be useful for various applications.

Another interesting open problem is whether the ideas developed during the design of AC can also lead to applications in classical information theory.

Experimental Results

Various ideas of AC had already led to several experiments using 3–4 qubit quantum computing devices: 1. An experiment [4] that implemented a single RPC step. 2. An experiment [3] in which entropy-conservation bounds (which apply in any closed system) were bypassed. 3. A full AC experiment [1] that includes the initialization of three carbon nuclei to the bias of a hydrogen spin, followed by a single compression step on these three carbons.

Cross References

- ► Dictionary-Based Data Compression
- ► Quantum Algorithm for Factoring
- ► Quantum Algorithm for the Parity Problem
- ► Quantum Dense Coding
- ► Quantum Key Distribution

Recommended Reading

1. Baugh, J., Moussa, O., Ryan, C.A., Nayak, A., Laflamme, R.: Experimental implementation of heat-bath algorithmic cooling using solid-state nuclear magnetic resonance. Nature **438**, 470–473 (2005)
2. Boykin, P.O., Mor, T., Roychowdhury, V., Vatan, F., Vrijen, R.: Algorithmic cooling and scalable NMR quantum computers. Proc. Natl. Acad. Sci. **99**, 3388–3393 (2002)
3. Brassard, G., Elias, Y., Fernandez, J.M., Gilboa, H., Jones, J.A., Mor, T., Weinstein, Y., Xiao, L.: Experimental heat-bath cooling of spins. Submitted to Proc. Natl. Acad. Sci. USA. See also quant-ph/0511156 (2005)
4. Chang, D.E., Vandersypen, L.M.K., Steffen, M.: NMR implementation of a building block for scalable quantum computation. Chem. Phys. Lett. **338**, 337–344 (2001)
5. Cory, D.G., Fahmy, A.F., Havel, T.F.: Ensemble quantum computing by NMR spectroscopy. Proc. Natl. Acad. Sci. **94**, 1634–1639 (1997)
6. Elias, Y., Fernandez, J.M., Mor, T., Weinstein, Y.: Optimal algorithmic cooling of spins. Isr. J. Chem. **46**, 371–391 (2006), also in: Ekl, S. et al. (eds.) Lecture Notes in Computer Science, Volume 4618, pp. 2–26. Springer, Berlin (2007), Unconventional Computation. Proceedings of the Sixth International Conference UC2007 Kingston, August 2007
7. Fernandez, J.M.: De computatione quantica. Dissertation, University of Montreal (2004)
8. Fernandez, J.M., Lloyd, S., Mor, T., Roychowdhury V.: Practicable algorithmic cooling of spins. Int. J. Quant. Inf. **2**, 461–477 (2004)
9. Gershenfeld, N.A., Chuang, I.L.: Bulk spin-resonance quantum computation. Science **275**, 350–356 (1997)
10. Mor, T., Roychowdhury, V., Lloyd, S., Fernandez, J.M., Weinstein, Y.: Algorithmic cooling. US Patent 6,873,154 (2005)
11. Schulman, L.J., Mor, T., Weinstein, Y.: Physical limits of heat-bath algorithmic cooling. Phys. Rev. Lett. **94**, 120501, pp. 1–4 (2005)
12. Schulman, L.J., Mor, T., Weinstein, Y.: Physical limits of heat-bath algorithmic cooling. SIAM J. Comput. **36**, 1729–1747 (2007)
13. Schulman, L.J., Vazirani, U.: Molecular scale heat engines and scalable quantum computation. Proc. 31st ACM STOC, Symp. Theory of Computing,pp. 322–329 Atlanta, 01–04 May 1999
14. Sørensen, O.W.: Polarization transfer experiments in high-resolution NMR spectroscopy. Prog. Nuc. Mag. Res. Spect. **21**, 503–569 (1989)

Algorithmic Mechanism Design

1999; Nisan, Ronen

RON LAVI
Faculty of Industrial Engineering and Management, Technion, Haifa, Israel

Problem Definition

Mechanism design is a sub-field of economics and game theory that studies the construction of social mechanisms in the presence of selfish agents. The nature of the agents dictates a basic contrast between the social planner, that aims to reach a socially desirable outcome, and the agents, that care only about their own private utility. The underlying question is how to incentivize the agents to cooperate, in order to reach the desirable social outcomes.

In the Internet era, where computers act and interact on behalf of selfish entities, the connection of the above to algorithmic design suggests itself: suppose that the input to an algorithm is kept by selfish agents, who aim to maximize their own utility. How can one design the algorithm so that the agents will find it in their best interest to cooperate, and a close-to-optimal outcome will be outputted? This is different than classic distributed computing models, where agents are either "good" (meaning obedient) or "bad" (meaning faulty, or malicious, depending on the context). Here, no such partition is possible. It is simply assumed that all agents are utility maximizers. To illustrate this, let us describe a motivating example:

A Motivating Example: Shortest Paths

Given a weighted graph, the goal is to find a shortest path (with respect to the edge weights) between a given source and target nodes. Each edge is controlled by a selfish entity, and the weight of the edge, w_e is private information of that edge. If an edge is chosen by the algorithm to be included in the shortest path, it will incur a cost which is minus its weight (the cost of communication). Payments to the edges are allowed, and the total utility of an edge that participates in the shortest path and gets a payment p_e is assumed to be $u_e = p_e - w_e$. Notice that the shortest path is *with respect to the true weights of the agents, although these are not known to the designer.*

Assuming that each edge will act in order to maximize its utility, how can one choose the path and the payments? One option is to ignore the strategic issue all together, ask the edges to simply report their weights, and compute the shortest path. In this case, however, an edge dislikes being selected, and will therefore prefer to report a very high weight (much higher than its true weight) in order to decrease the chances of being selected. Another option is to pay each selected edge its reported weight, or its reported weight plus a small fixed "bonus". However in such a case all edges will report lower weights, as being selected will imply a positive gain.

Although this example is written in an algorithmic language, it is actually a mechanism design problem, and the solution, which is now a classic, was suggested in the 70's. The chapter continues as follows: First, the abstract formulation for such problems is given, the classic solution from economics is described, and its advantages and disadvantages for algorithmic purposes are discussed. The next section then describes the new results that algorithmic mechanism design offers.

Abstract Formulation

The framework consists of a set A of alternatives, or outcomes, and n players, or agents. Each player i has a valuation function $v_i : A \rightarrow \Re$ that assigns a value to each possible alternative. This valuation function belongs to a domain V_i of all possible valuation functions. Let $V = V_1 \times \cdots \times V_n$, and $V_{-i} = \prod_{j \neq i} V_j$. Observe that this generalizes the shortest path example of above: A is all the possible $s - t$ paths in the given graph, $v_e(a)$ for some path $a \in A$ is either $-w_e$ (if $e \in a$) or zero.

A *social choice function* $f : V \rightarrow A$ assigns a socially desirable alternative to any given profile of players' valuations. This parallels the notion of an algorithm. A *mechanism* is a tuple $M = (f, p_1, \ldots, p_n)$, where f is a social choice function, and $p_i : V \rightarrow \Re$ (for $i = 1, \ldots, n$) is the price charged from player i. The interpretation is that the social planner asks the players to reveal their true valuations, chooses the alternative according to f as if the players have indeed acted truthfully, and in addition rewards/punishes the players with the prices. These prices should induce "truthfulness" in the following strong sense: no matter what the other players declare, it is always in the best interest of player i to reveal her true valuation, as this will maximize her utility. Formally, this translates to:

Definition 1 (Truthfulness) M is "truthful" (in dominant strategies) if, for any player i, any profile of valuations of the other players $v_{-i} \in V_{-i}$, and any two valuations of player $i v_i, v'_i \in V_i$,

$$v_i(a) - p_i(v_i, v_{-i}) \geq v_i(b) - p_i(v'_i, v_{-i})$$

where $f(v_i, v_{-i}) = a$ and $f(v'_i, v_{-i}) = b$.

Truthfulness is quite strong: a player need not know anything about the other players, even not that they are rational, and still determine the best strategy for her. Quite remarkably, there exists a truthful mechanism, even under the current level of abstraction. This mechanism suits all problem domains, where the social goal is to maximize the "social welfare":

Definition 2 (Social welfare maximization) A social choice function $f : V \rightarrow A$ maximizes the social welfare if $f(v) \in \operatorname{argmax}_{a \in A} \sum_i v_i(a)$, for any $v \in V$.

Notice that the social goal in the shortest path domain is indeed welfare maximization, and, in general, this is a natural and important economic goal. Quite remarkably, there exists a general technique to construct truthful mechanisms that implement this goal:

Theorem 1 (Vickrey–Clarke–Groves (VCG)) *Fix any alternatives set A and any domain V, and suppose that $f : V \rightarrow A$ maximizes the social welfare. Then there exist prices p such that the mechanism (f, p) is truthful.*

This gives "for free" a solution to the shortest path problem, and to many other algorithmic problems. The great advantage of the VCG scheme is its generality: it suits *all* problem domains. The disadvantage, however, is that the method is tailored to social welfare maximization. This turns out to be restrictive, especially for algorithmic and computational settings, due to several reasons: (i) different algorithmic goals: the algorithmic literature considers a variety of goals, including many that cannot be translated to welfare maximization. VCG does not help us in such cases. (ii) computational complexity: even if the goal is welfare maximization, in many settings achieving exactly the optimum is computationally hard. The CS discipline usually overcomes this by using approximation algorithms, but VCG will not work with such algorithm – reaching exact optimality is a necessary requirement of VCG. (iii) different algorithmic models: common CS models change "the basic setup", hence cause unexpected difficulties when one tries to use VCG (for example, an online model, where the input is revealed over time; this is common in CS, but changes the implicit setting that VCG requires). This is true even if welfare maximization is still the goal.

Answering any one of these difficulties requires the design of a non-VCG mechanism. What analysis tools should be used for this purpose? In economics and classic mechanism design, average-case analysis, that relies on the knowledge of the underlying distribution, is the standard. Computer science, on the other hand, usually prefers to avoid strong distributional assumptions, and to use worst-case analysis. This difference is another cause to the uniqueness of the answers provided by algorithmic mechanism design. Some of the new results that have emerged as a consequence of this integration between Computer Science and Economics is next described. Many other research topics that use the tools of algorithmic mechanism design are described in the entries on Adword Pricing, Competitive Auctions, False Name Proof Auctions, Generalized Vickrey Auction, Incentive Compatible Ranking, Mechanism for One Parameter Agents Single Buyer/Seller, Multiple Item Auctions, Position Auctions, and Truthful Multicast.

There are two different but closely related research topics that should be mentioned in the context of this entry. The first is the line of works that studies the "price of anarchy" of a given system. These works analyze *existing* systems, trying to quantify the loss of social efficiency due to the selfish nature of the participants, while the approach of algorithmic mechanism design is to understand how new systems should be designed. For more details on this topic the reader is referred to the entry on Price of Anarchy. The second topic regards the algorithmic study of various equilibria computation. These works bring computational aspects into economics and game theory, as they ask what equilibria notions are reasonable to assume, if one requires computational efficiency, while the works described here bring game theory and economics into computer science and algorithmic theory, as they ask what algorithms are reasonable to design, if one requires the resilience to selfish behavior. For more details on this topic the reader is referred (for example) to the entry on Algorithms for Nash Equilibrium and to the entry on General Equilibrium.

Key Results

Problem Domain 1: Job Scheduling

Job scheduling is a classic algorithmic setting: n jobs are to be assigned to m machines, where job j requires processing time p_{ij} on machine i. In the game-theoretic setting, it is assumed that each machine i is a selfish entity, that incurs a cost p_{ij} from processing job j. Note that the payments in this setting (and in general) may be negative, offsetting such costs. A popular algorithmic goal is to assign jobs to machines in order to minimize the "makespan": $\max_i \sum_{j \text{ is assigned to } i} p_{ij}$. This is different than welfare maximization, which translates in this setting to the minimization of $\sum_i \sum_{j \text{ is assigned to } i} p_{ij}$, further illustrating the problem of different algorithmic goals. Thus the VCG scheme cannot be used, and new methods must be developed.

Results for this problem domain depend on the specific assumptions about the structure of the processing time vectors. In the *related machines* case, $p_{ij} = p_j/s_i$ for any ij, where the p_j's are public knowledge, and the only secret parameter of player i is its *speed*, s_i.

Theorem 2 ([3,22]) *For job scheduling on related machines, there exists a truthful exponential-time mechanism that obtains the optimal makespan, and a truthful polynomial-time mechanism that obtains a 3-approximation to the optimal makespan.*

More details on this result are given in the entry on Mechanism for One Parameter Agents Single Buyer. The bottom line conclusion is that, although the social goal is different than welfare maximization, there still exists a truthful mechanism for this goal. A non-trivial approximation guarantee is achieved, even under the additional requirement of computational efficiency. However, this guarantee does not match the best possible without the truthfulness requirement, since in this case a PTAS is known.

Open Question 1 *Is there a truthful PTAS for makespan minimization in related machines?*

If the number of machines is fixed then [2] give such a truthful PTAS.

The above picture completely changes in the move to the more general case of *unrelated machines*, where the p_{ij}'s are allowed to be arbitrary:

Theorem 3 ([13,30]) *Any truthful scheduling mechanism for unrelated machines cannot approximate the optimal makespan by a factor better than $1 + \sqrt{2}$ (for deterministic mechanisms) and $2 - 1/m$ (for randomized mechanisms).*

Note that this holds regardless of computational considerations. In this case, switching from welfare maximization to makespan minimization results in a strong impossibility. On the possibilities side, virtually nothing (!) is known. The VCG mechanism (which minimizes the total social cost) is an m-approximation of the optimal makespan [32], and, in fact, nothing better is currently known:

Open Question 2 *What is the best possible approximation for truthful makespan minimization in unrelated machines?*

What caused the switch from "mostly possibilities" to "mostly impossibilities"? Related machines is a single-dimensional domain (players hold only one secret number), for which truthfulness is characterized by a simple monotonicity condition, that leaves ample flexibility for algorithmic design. Unrelated machines, on the other hand, are a multi-dimensional domain, and the algorithmic conditions implied by truthfulness in such a case are harder to work with. It is still unclear whether these conditions imply real mathematical impossibilities, or perhaps just pose harder obstacles that can be in principle solved. One multi-dimensional scheduling domain for which possibility results are known is the case where $p_{ij} \in \{L_j, H_j\}$, where the "low" 's and "high" 's are fixed and known. This case generalizes the classic multi-dimensional model of restricted machines ($p_{ij} \in \{p_j, \infty\}$), and admits a truthful 3-approximation [27].

Problem Domain 2: Digital Goods and Revenue Maximization

In the E-commerce era, a new kind of "digital goods" have evolved: goods with no marginal production cost, or, in other words, goods with unlimited supply. One example is songs being sold on the Internet. There is a sunk cost of producing the song, but after that, additional electronic copies incur no additional cost. How should such items be sold? One possibility is to conduct an *auction*. An auction is a one-sided market, where a monopolistic entity (the auctioneer) wishes to sell one or more items to a set of buyers.

In this setting, each buyer has a privately known value for obtaining one copy of the good. Welfare maximization simply implies the allocation of one good to every buyer, but a more interesting question is the question of revenue maximization. How should the auctioneer design the auction in order to maximize his profit? Standard tools from the study of revenue-maximizing auctions[1] suggest to simply declare a price-per-buyer, determined by the probability distribution of the buyer's value, and make a take-it-or-leave-it offer. However, such a mechanism needs to know the underlying distribution. Algorithmic mechanism design suggests an alternative, worst-case result, in the spirit of CS-type models and analysis.

[1]This model was not explicitly studied in classic auction theory, but standard results from there can be easily adjusted to this setting.

Suppose that the auctioneer is required to sell all items in the same price, as is the case for many "real-life" monopolists, and denote by $F(\vec{v})$ the maximal revenue from a fixed-price sale to bidders with values $\vec{v} = v_1, \dots v_n$, *assuming that all values are known*. Reordering indexes so that $v_1 \geq v_2 \geq \cdots \geq v_n$, let $F(\vec{v}) = \max_i i \cdot v_i$. The problem is, of-course, that in fact *nothing* about the values is known. Therefore, a truthful auction that extracts the players' values is in place. Can such an auction obtain a profit that is a constant fraction of $F(\vec{v})$, for any $\vec{v}$ (i. e. in the worst case)? Unfortunately, the answer is provably no [17]. The proof makes use of situations where the entire profit comes from the highest bidder. Since there is no potential for competition among bidders, a truthful auction cannot force this single bidder to reveal her value.

Luckily, a small relaxation in the optimality criteria significantly helps. Specifically, denote by $F^{(2)}(\vec{v}) = \max_{i \geq 2} i \cdot v_i$ (i. e. the benchmark is the auction that sells to at least two buyers).

Theorem 4 ([17,20]) *There exists a truthful randomized auction that obtains an expected revenue of at least $F^{(2)}/3.25$, even in the worst-case. On the other hand, no truthful auction can approximate $F^{(2)}$ within a factor better than 2.42.*

Several interesting formats of distribution-free revenue-maximizing auctions have been considered in the literature. The common building block in all of them is the random partitioning of the set of buyers to random subsets, analyzing each set separately, and using the results on the other sets. Each auction utilizes a different analysis on the two subsets, which yields slightly different approximation guarantees. [1] describe an elegant method to derandomize these type of auctions, while losing another factor of 4 in the approximation. More details on this problem domain can be found in the entry on Competitive Auctions.

Problem Domain 3: Combinatorial Auctions

Combinatorial auctions (CAs) are a central model with theoretical importance and practical relevance. It generalizes many theoretical algorithmic settings, like job scheduling and network routing, and is evident in many real-life situations. This new model has various pure computational aspects, and, additionally, exhibits interesting game theoretic challenges. While each aspect is important on its own, obviously only the integration of the two provides an acceptable solution.

A combinatorial auction is a multi-item auction in which players are interested in *bundles* of items. Such a valuation structure can represent substitutabilities among items, complementarities among items, or a combination of both. More formally, m items (Ω) are to be allocated to n players. Players value subsets of items, and $v_i(S)$ denotes i's value of a bundle $S \subseteq \Omega$. Valuations additionally satisfy: (i) monotonicity, i.e $v_i(S) \le v_i(T)$ for $S \subseteq T$, and (ii) normalization, i. e. $v_i(\emptyset) = 0$. The literature has mostly considered the goal of maximizing the social welfare: find an allocation $(S_1, \ldots, S_n)$ that maximizes $\sum_i v_i(S_i)$.

Since a general valuation has size exponential in n and m, the representation issue must be taken into account. Two models are usually considered (see [11] for more details). In the *bidding languages* model, the bid of a player represents his valuation is a concise way. For this model it is NP-hard to approximate the social welfare within a ratio of $\Omega(m^{1/2-\epsilon})$, for any $\epsilon > 0$ (if "single-minded" bids are allowed; the exact definition is given below). In the *query access* model, the mechanism iteratively queries the players in the course of computation. For this model, any algorithm with polynomial communication cannot obtain an approximation ratio of $\Omega(m^{1/2-\epsilon})$ for any $\epsilon > 0$. These bounds are tight, as there exist a deterministic $\sqrt{m}$-approximation with polynomial computation and communication. Thus, for the general valuation structure, the computational status by itself is well-understood.

The basic incentives issue is again well-understood: VCG obtains truthfulness. Since VCG requires the exact optimum, which is NP-hard to compute, the two considerations therefore clash, when attempting to use classic techniques. Algorithmic mechanism design aims to develop new techniques, to integrate these two desirable aspects.

The first positive result for this integration challenge was given by [29], for the special case of "single-minded bidders": each bidder, i, is interested in a specific bundle S_i, for a value v_i (any bundle that contains S_i is worth v_i, and other bundles have zero value). Both v_i, S_i are private to the player i.

Theorem 5 ([29]) *There exists a truthful and polynomial-time deterministic combinatorial auction for single-minded bidders, which obtains a $\sqrt{m}$-approximation to the optimal social welfare.*

A possible generalization of the basic model is to assume that each item has B copies, and each player still desires at most one copy from each item. This is termed "multi-unit CA". As B grows, the integrality constraint of the problem reduces, and so one could hope for better solutions. Indeed, the next result exploits this idea:

Theorem 6 ([7]) *There exists a truthful and polynomial-time deterministic multi-unit CA, for $B \ge 3$ copies of each item, that obtains $O(B \cdot m^{1/(B-2)})$-approximation to the optimal social welfare.*

This auction copes with the representation issue (since general valuations are assumed) by accessing the valuations through a "demand oracle": given per-item prices $\{p_x\}_{x \in \Omega}$, specify a bundle S that maximizes $v_i(S) - \sum_{x \in S} p_x$.

Two main drawbacks of this auction motivate further research on the issue. First, as B gets larger it is reasonable to expect the approximation to approach 1 (indeed polynomial-time algorithms with such an approximation guarantee do exist). However here the approximation ratio does not decrease below $O(\log m)$ (this ratio is achieved for $B = O(\log m)$). Second, this auction does not provide a solution to the original setting, where $B = 1$, and, in general for small B's the approximation factor is rather high. One way to cope with these problems is to introduce randomness:

Theorem 7 ([26]) *There exists a truthful-in-expectation and polynomial-time randomized multi-unit CA, for any $B \ge 1$ copies of each item, that obtains $O(m^{1/(B+1)})$-approximation to the optimal social welfare.*

Thus, by allowing randomness, the gap from the standard computational status is being completely closed. The definition of truthfulness-in-expectation is the natural extension of truthfulness to a randomized environment: the *expected* utility of a player is maximized by being truthful.

However, this notion is strictly weaker than the deterministic notion, as this implicitly implies that players care only about the expectation of their utility (and not, for example, about the variance). This is termed "the risk-neutrality" assumption in the economics literature. An intermediate notion for randomized mechanisms is that of "universal truthfulness": the mechanism is truthful given any fixed result of the coin toss. Here, risk-neutrality is no longer needed. [15] give a universally truthful CA for $B = 1$ that obtains an $O(\sqrt{m})$-approximation. Universally truthful mechanisms are still weaker than deterministic truthful mechanisms, due to two reasons: (i) It is not clear how to actually create the correct and exact probability distribution with a deterministic computer. The situation here is different than in "regular" algorithmic settings, where various derandomization techniques can be employed, since these in general does not carry through the truthfulness property. (ii) Even if a natural randomness source exists, one cannot improve the quality of the actual output by repeating the computation several times (using the the law of large numbers). Such a repetition will again destroy truthfulness. Thus, exactly because the game-theoretic issues are being considered in parallel to the computational ones, the importance of determinism increases.

Open Question 3 *What is the best-possible approximation ratio that deterministic and truthful combinatorial auctions can obtain, in polynomial-time?*

There are many valuation classes, that restrict the possible valuations to some reasonable format (see [28] for more details). For example, sub-additive valuations are such that, for any two bundles $S, T, \subseteq \Omega$, $v(S \cup T) \leq v(S) + v(T)$. Such classes exhibit much better approximation guarantees, e. g. for sub-additive valuation a polynomial-time 2-approximation is known [16]. However, no polynomial-time truthful mechanism (be it randomized, or deterministic) with a constant approximation ratio, is known for any of these classes.

Open Question 4 *Does there exist polynomial-time truthful constant-factor approximations for special cases of CAs that are NP-hard?*

Revenue maximization in CAs is of-course another important goal. This topic is still mostly unexplored, with few exceptions. The mechanism [7] obtains the same guarantees with respect to the optimal revenue. Improved approximations exist for multi-unit auctions (where all items are identical) with budget constrained players [12], and for unlimited-supply CAs with single-minded bidders [6].

The topic of Combinatorial Auctions is discussed also in the entry on Multiple Item Auctions.

Problem Domain 4: Online Auctions

In the classic CS setting of "online computation", the input to an algorithm is not revealed all at once, before the computation begins, but gradually, over time (for a detailed discussion see the many entries on online problems in this book). This structure suits the auction world, especially in the new electronic environments. What happens when players arrive over time, and the auctioneer must make decisions facing only a subset of the players at any given time?

The integration of online settings, worst-case analysis, and auction theory, was suggested by [24]. They considered the case where players arrive one at a time, and the auctioneer must provide an answer to each player *as it arrives*, without knowing the future bids. There are k identical items, and each bidder may have a distinct value for every possible quantity of the item. These values are assumed to be marginally decreasing, where each marginal value lies in the interval $[\underline{v}, \bar{v}]$. The private information of a bidder includes both her valuation function, and her arrival time, and so a truthful auction need to incentivize the players to arrive on time (and not later on), and to reveal their true values. The most interesting result in this setting is for a large k, so that in fact there is a continuum of items:

Theorem 8 **([24])** *There exists a truthful online auction that simultaneously approximates, within a factor of* $O(\log(\bar{v}/\underline{v}))$*, the optimal offline welfare, and the offline revenue of VCG. Furthermore, no truthful online auction can obtain a better approximation ratio to either one of these criteria (separately).*

This auction has the interesting property of being a "posted price" auction. Each bidder is not required to reveal his valuation function, but, rather, he is given a price for each possible quantity, and then simply reports the desired quantity under these prices.

Ideas from this construction were later used by [10] to construct *two*-sided online auction markets, where multiple sellers and buyers arrive online.

This approximation ratio can be dramatically improved, to be a constant, 4, if one assumes that (i) there is only one item, and (ii) player values are i.i.d from some fixed distribution. No a–priori knowledge of this distribution is needed, as neither the mechanism nor the players are required to make any use of it. This work, [19], analyzes this by making an interesting connection to the class of "secretary problems".

A general method to convert online algorithms to online mechanisms is given by [4]. This is done for one item auctions, and, more generally, for one parameter domains. This method is competitive both with respect to the welfare and the revenue.

The revenue that the online auction of Theorem 8 manages to raise is competitive only with respect to VCG's revenue, which may be far from optimal. A parallel line of works is concerned with revenue maximizing auctions. To achieve good results, two assumptions need to be made: (i) there exists an unlimited supply of items (and recall from Sect. "Problem Domain 2: Digital Goods and Revenue Maximization" that $F(v)$ is the offline optimal monopolistic fixed-price revenue), and (ii) players cannot lie about their arrival time, only about their value. This last assumption is very strong, but apparently needed. Such auctions are termed here "value-truthful", indicating that "time-truthfulness" is missing.

Theorem 9 ([9]) *For any $\epsilon > 0$, there exists a value-truthful online auction, for the unlimited supply case, with expected revenue of at least $(F(v))/(1+\epsilon) - O(h/\epsilon^2)$.*

The construction exploits principles from learning theory in an elegant way. Posted price auctions for this case are also possible, in which case the additive loss increases to $O(h \log\log h)$. [19] consider fully-truthful online auctions for revenue maximization, but manage to obtain only very high (although fixed) competitive ratios. Constructing fully-truthful online auctions with a close-to-optimal revenue remains an open question. Another interesting open question involves multi-dimensional valuations. The work [24] remains the only work for players that may demand multiple items. However their competitive guarantees are quite high, and achieving better approximation guarantees (especially with respect to the revenue) is a challenging task.

Advanced Issues

Monotonicity What is the general way for designing a truthful mechanism? The straight-forward way is to check, for a given social choice function f, whether truthful prices exist. If not, try to "fix" f. It turns out, however, that there exists a more structured way, an *algorithmic* condition that will imply the *existence* of truthful prices. Such a condition shifts the designer back to the familiar territory of algorithmic design. Luckily, such a condition do exist, and is best described in the abstract social choice setting of Sect. "Problem Definition":

Definition 3 ([8,23]) A social choice function $f: V \to A$ is "weakly monotone" (W-MON) if for any i, $v_{-i} \in V_{-i}$, and any $v_i, v_i' \in V_i$, the following holds. Suppose that $f(v_i, v_{-i}) = a$, and $f(v_i', v_{-i}) = b$. Then $v_i'(b) - v_i(b) \geq v_i'(a) - v_i(a)$.

In words, this condition states the following. Suppose that player i changes her declaration from v_i to v_i', and this causes the social choice to change from a to b. Then it must be the case that i's value for b has increased in the transition from v_i to v_i' no-less than i's value for a.

Theorem 10 ([35]) *Fix a social choice function $f: V \to A$, where V is convex, and A is finite. Then there exist prices p such that $M = (f, p)$ is truthful if and only if f is weakly monotone.*

Furthermore, given a weakly monotone f, there exists an explicit way to determine the appropriate prices p (see [18] for details).

Thus, the designer should aim for weakly monotone algorithms, and need not worry about actual prices. But how difficult is this? For single-dimensional domains, it turns out that W-MON leaves ample flexibility for the algorithm designer. Consider for example the case where every alternative has a value of either 0 (the player "loses") or some $v_i \in \Re$ (the player "wins" and obtains a value v_i). In such a case, it is not hard to show that W-MON reduces to the following monotonicity condition: if a player wins with v_i, and increases her value to $v_i' > v_i$ (while v_{-i} remains fixed), then she must win with v_i' as well. Furthermore, in such a case, the price of a winning player must be set to the infimum over all winning values.

Impossibilities of truthful design It is fairly simple to construct algorithms that satisfy W-MON for single-dimensional domains, and a variety of positive results were obtained for such domains, in classic mechanism design, as well as in algorithmic mechanism design. But how hard is it to satisfy W-MON for multi-dimensional domains? This question is yet unclear, and seems to be one of the challenges of algorithmic mechanism design. The contrast between single-dimensionality and multi-dimensionality appears in all problem domains that were surveyed here, and seems to reflect some inherent difficulty that is not exactly understood yet. Given a social choice function f, call f *implementable* (in dominant strategies) if there exist prices p such that $M = (f, p)$ is truthful. The basic question is then *what forms of social choice functions are implementable.*

As detailed in the beginning, the welfare maximizing social choice function is implementable. This specific function can be slightly generalized to allow weights, in the following way: fix some non-negative real constants $\{w_i\}_{i=1}^n$ (not all are zero) and $\{\gamma_a\}_{a \in A}$, and choose an alternative that maximizes the *weighted* social welfare, i. e. $f(v) \in \text{argmax}_{a \in A} \sum_i w_i v_i(a) + \gamma_a$. This class of functions is sometimes termed "affine maximizers". It turns out that these functions are also implementable, with prices similar in spirit to VCG. In the context of the above characterization question, one sharp result stands out:

Theorem 11 ([34]) *Fix a social choice function $f: V \to A$, such that (i) A is finite, $|A| \geq 3$, and f is onto A, and (ii) $V_i = \Re^A$ for all i. Then f is implementable (in dominant strategies) if and only if it is an affine maximizer.*

The domain V that satisfies $V_i = \Re^A$ for all i is term an "unrestricted domain". The theorem states that, if the domain is unrestricted, at least three alternatives are chosen, and the set A of alternatives is finite, then nothing besides affine maximizers can be implemented!

However, the assumption that the domain is unrestricted is very restrictive. All the above example domains exhibit some basic combinatorial structure, and are therefore restricted in some way. And as discussed above, for many restricted domains the theorem is simply not true. So what *is* the possibilities – impossibilities border? As mentioned above, this is an unsolved challenge. Lavi, Mu'alem, and Nisan [23] explore this question for Combinatorial Auctions and similar restricted domains, and reach partial answers. For example:

Theorem 12 ([23]) *Any truthful combinatorial auction or multi-unit auction among two players, that must always allocate all items, and that approximates the welfare by a factor better than 2, must be an affine maximizer.*

Of-course, this is far from being a complete answer. What happens if there are more than two players? And what happens if it is possible to "throw away" part of the items? These questions, and the more general and abstract characterization question, are all still open.

Alternative solution concepts In light of the conclusions of the previous section, a natural thought would be to re-examine the *solution concept* that is being used. Truthfulness relies on the strong concept of dominant strategies: for each player there is a unique strategy that maximizes her utility, no matter what the other players are doing. This is very strong, but it fits very well the worst-case way of thinking in CS. What other solution concepts can be used? As described above, randomization, and truthfulness-in-expectation, can help. A related concept, again for randomized mechanisms, is truthfulness with high probability. Another direction is to consider mechanisms where players cannot improve their utility *too much* by deviating from the truth-telling strategy [21].

Algorithm designers do not care so much about actually reaching an equilibrium point, or finding out what will the players play – the major concern is to guarantee the optimality of the solution, taking into account the strategic behavior of the players. Indeed, one way of doing this is to guarantee a good equilibrium point. But there is no reason to rule out mechanisms where several acceptable strategic choices for the players exist, provided that the approximation will be achieved *in each of these choices.*

As a first attempt, one is tempted to simply let the players try and improve the basic result by allowing them to lie. However, this can cause unexpected dynamics, as each player chooses her lies under some assumptions about the lies of the others, etc. etc. To avoid such an unpredictable situation, it is important to insist on using rigorous game theoretic reasoning to explain exactly why the outcome will be satisfactory.

The work [31] suggests the notion of "feasibly dominant" strategies, where players reveal the possible lies they consider, and the mechanism takes this into account. By assuming that the players are computationally bounded, one can show that, instead of actually "lying", the players will prefer to reveal their true types plus all the lies they might consider. In such a case, since the mechanism has obtained the true types of the players, a close-to-optimal outcome will be guaranteed.

Another definition tries to capture the initial intuition by using the classic game-theoretic notion of undominated strategies:

Definition 4 ([5]) A mechanism M is an "algorithmic implementation of a c-approximation (in undominated strategies)" if there exists a set of strategies, D, such that (i) M obtains a c-approximation for any combination of strategies from D, in polynomial time, and (ii) For any strategy not in D, there exists a strategy in D that weakly dominates it, and this transition is polynomial-time computable.

By the second condition, it is reasonable to assume that a player will indeed play *some* strategy in D, and, by the first condition, it does not matter what tuple of strategies in D will actually be chosen, as any of these will provide the approximation. This transfers some of the burden from the game-theoretic design to the algorithmic design, since now a guarantee on the approximation should bu provided for a larger range of strategies. [5] exploit this notion to design a deterministic CA for multi-dimensional players that achieves a close-to-optimal approximation guarantee. A similar-in-spirit notion, although a weaker one, is the notion of "Set-Nash" [25].

Applications

One of the popular examples to a "real-life" combinatorial auction is the spectrum auction that the US government conducts, in order to sell spectrum licenses. Typical bids reflect values for different spectrum ranges, to accommodate different geographical and physical needs, where different spectrum ranges may complement or substitute one another. The US government invests research efforts in order to determine the best format for such an auction, and auction theory is heavily exploited. Interestingly, the US law guides the authorities to allocate these spectrum ranges in a way that will maximize *the social welfare*, thus providing a good example for the usefulness of this goal.

Adword auctions are another new and fast-growing application of auction theory in general, and of the new algorithmic auctions in particular. These are auctions that determine the advertisements that web-search engines place close to the search results they show, after the user submits her search keywords. The interested companies compete, for every given keyword, on the right to place their ad on the results' page, and this turns out to be the main source of income for companies like Google. Several entries in this book touch on this topic in more details, including the entries on Adwords Pricing and on Position Auctions.

A third example to a possible application, in the meanwhile implemented only in the academic research labs, is the application of algorithmic mechanism design to pricing and congestion control in communication networks. The existing fixed pricing scheme has many disadvantages, both with respect to the needs of efficiently allocating the available resources, and with respect to the new opportunities of the Internet companies to raise more revenue due to specific types of traffic. Theory suggests solutions to both of these problems.

Cross References

► Adwords Pricing
► Competitive Auction
► False-Name-Proof Auction
► Generalized Vickrey Auction
► Incentive Compatible Selection
► Position Auction
► Truthful Multicast

Recommended Reading

The topics presented here are detailed in the textbook [33]. Section "Problem Definition" is based on the paper [32], that also coined the term "algorithmic mechanism design". The book [14] covers the various aspects of combinatorial auctions.

1. Aggarwal, G., Fiat, A., Goldberg, A., Immorlica, N., Sudan, M.: Derandomization of auctions. In: Proc. of the 37th ACM Symposium on Theory of Computing (STOC'05), 2005
2. Andelman, N., Azar, Y., Sorani, M.: Truthful approximation mechanisms for scheduling selfish related machines. In: Proc. of the 22nd International Symposium on Theoretical Aspects of Computer Science (STACS), 2005, pp. 69–82
3. Archer, A., Tardos, É.: Truthful mechanisms for one-parameter agents. In: Proc. 42nd Annual Symposium on Foundations of Computer Science (FOCS), 2001, pp. 482–491
4. Awerbuch, B., Azar, Y., Meyerson, A.: Reducing truth-telling online mechanisms to online optimization. In: Proc. of the 35th ACM Symposium on Theory of Computing (STOC'03), 2003
5. Babaioff, M., Lavi, R., Pavlov, E.: Single-value combinatorial auctions and implementation in undominated strategies. In: Proc. of the 17th Symposium on Discrete Algorithms (SODA), 2006
6. Balcan, M., Blum, A., Hartline, J., Mansour, Y.: Mechanism design via machine learning. In: Proc. of the 46th Annual Symposium on Foundations of Computer Science (FOCS'05), 2005
7. Bartal, Y., Gonen, R., Nisan, N.: Incentive compatible multi-unit combinatorial auctions. In: Proc. of the 9th Conference on Theoretical Aspects of Rationality and Knowledge (TARK'03), 2003
8. Bikhchandani, S., Chatterjee, S., Lavi, R., Mu'alem, A., Nisan, N., Sen, A.: Weak monotonicity characterizes deterministic dominant-strategy implementation. Econometrica **74**, 1109–1132 (2006)
9. Blum, A., Hartline, J.: Near-optimal online auctions. In: Proc. of the 16th Symposium on Discrete Algorithms (SODA), 2005
10. Blum, A., Sandholm, T., Zinkevich, M.: Online algorithms for market clearing. J. ACM **53**(5), 845–879 (2006)
11. Blumrosen, L., Nisan, N.: On the computational power of iterative auctions. In: Proc. of the 7th ACM Conference on Electronic Commerce (EC'05), 2005
12. Borgs, C., Chayes, J., Immorlica, N., Mahdian, M., Saberi, A.: Multi-unit auctions with budget-constrained bidders. In: Proc. of the 7th ACM Conference on Electronic Commerce (EC'05), 2005
13. Christodoulou, G., Koutsoupias, E., Vidali, A.: A lower bound for scheduling mechanisms. In: Proc. 18th Symposium on Discrete Algorithms (SODA), 2007
14. Cramton, P., Shoham, Y., Steinberg, R.: Combinatorial Auctions. MIT Press (2005)
15. Dobzinski, S., Nisan, N., Schapira, M.: Truthful randomized mechanisms for combinatorial auctions. In: Proc. of the 38th ACM Symposium on Theory of Computing (STOC'06), 2006
16. Feige, U.: On maximizing welfare when utility functions are subadditive. In: Proc. of the 38th ACM Symposium on Theory of Computing (STOC'06), 2006
17. Goldberg, A., Hartline, J., Karlin, A., Saks, M., Wright, A.: Competitive auctions. Games Econ. Behav. **55**(2), 242–269 (2006)
18. Gui, H., Muller, R., Vohra, R.V.: Characterizing dominant strategy mechanisms with multi-dimensional types (2004). Working paper
19. Hajiaghayi, M., Kleinberg, R., Parkes, D.: Adaptive limited-supply online auctions. In: Proc. of the 6th ACM Conference on Electronic Commerce (EC'04), 2004
20. Hartline, J., McGrew, R.: From optimal limited to unlimited supply auctions. In: Proc. of the 7th ACM Conference on Electronic Commerce (EC'05), 2005
21. Kothari, A., Parkes, D., Suri, S.: Approximately-strategyproof and tractable multi-unit auctions. Decis. Support Syst. **39**, 105–121 (2005)
22. Kovács, A.: Fast monotone 3-approximation algorithm for scheduling related machines. In: Proc. 13th Annual European Symposium on Algorithms (ESA), 2005, pp. 616–627
23. Lavi, R., Mu'alem, A., Nisan, N.: Towards a characterization of truthful combinatorial auctions. In: Proc. of the 44rd Annual Symposium on Foundations of Computer Science (FOCS'03), 2003
24. Lavi, R., Nisan, N.: Competitive analysis of incentive compatible on-line auctions. Theor. Comput. Sci. **310**, 159–180 (2004)
25. Lavi, R., Nisan, N.: Online ascending auctions for gradually expiring items. In: Proc. of the 16th Symposium on Discrete Algorithms (SODA), 2005
26. Lavi, R., Swamy, C.: Truthful and near-optimal mechanism design via linear programming. In: Proc. 46th Annual Symposium

on Foundations of Computer Science (FOCS), 2005, pp. 595–604

27. Lavi, R., Swamy, C.: Truthful mechanism design for multi-dimensional scheduling via cycle monotonicity (2007). Working paper
28. Lehmann, B., Lehmann, D., Nisan, N.: Combinatorial auctions with decreasing marginal utilities. Games Econom. Behav. **55**(2), 270–296 (2006)
29. Lehmann, D., O'Callaghan, L., Shoham, Y.: Truth revelation in approximately efficient combinatorial auctions. J. ACM **49**(5), 577–602 (2002)
30. Mu'alem, A., Schapira, M.: Setting lower bounds on truthfulness. In: Proc. 18th Symposium on Discrete Algorithms (SODA), 2007
31. Nisan, N., Ronen, A.: Computationally feasible vcg mechanisms. In: Proc. of the 2nd ACM Conference on Electronic Commerce (EC'00), 2000
32. Nisan, N., Ronen, A.: Algorithmic mechanism design. Games Econom. Behav. **35**, 166–196 (2001)
33. Nisan, N., Roughgarden, T., Tardos, E., Vazirani, V.: Algorithmic Game Theory. Cambridge University Press (2007). (expected to appear)
34. Roberts, K.: The characterization of implementable choice rules. In: Laffont, J.J. (ed.) Aggregation and Revelation of Preferences, pp. 321–349. North-Holland (1979)
35. Saks, M., Yu, L.: Weak monotonicity suffices for truthfulness on convex domains. In: Proc. 6th ACM Conference on Electronic Commerce (ACM-EC), 2005, pp. 286–293

Algorithms for Spanners in Weighted Graphs

2003; Baswana, Sen

SURENDER BASWANA[1], SANDEEP SEN[2]
[1] Department of Computer Science and Engineering, IIT Kanpur, Kanpur, India
[2] Department of Computer Science and Engineering, IIT Delhi, New Delhi, India

Keywords and Synonyms

Graph algorithms; Randomized algorithms; Shortest path; Spanner

Problem Definition

A spanner is a *sparse* subgraph of a given undirected graph that preserves approximate distance between each pair of vertices. More precisely, a t-spanner of a graph $G = (V, E)$ is a subgraph (V, E_S), $E_S \subseteq E$ such that, for any pair of vertices, their distance in the subgraph is at most t times their distance in the original graph, where t is called the *stretch factor*. The spanners were defined formally by Peleg and Schäffer [14], though the associated notion was used implicitly by Awerbuch [3] in the context of network synchronizers.

Computing a t-spanner of smallest size for a given graph is a well motivated combinatorial problem with many applications. However, computing t-spanner of smallest size for a graph is NP-hard. In fact, for $t > 2$, it is NP-hard [10] even to approximate the smallest size of a t-spanner of a graph with ratio $O(2^{(1-\mu)\ln n})$ for any $\mu > 0$. Having realized this fact, researchers have pursued another direction which is quite interesting and useful. Let S_G^t be the size of the sparsest t-spanner of a graph G, and let S_n^t be the maximum value of S_G^t over all possible graphs on n vertices. Does there exist a polynomial time algorithm which computes, for any weighted graph and parameter t, its t-spanner of size $O(S_n^t)$? Such an algorithm would be the best one can hope for given the hardness of the original t-spanner problem. Naturally the question arises as to how large can S_n^t be? A 43-year old girth lower bound conjecture by Erdös [12] implies that there are graphs on n vertices whose $2k$- as well as $(2k-1)$-spanner will require $\Omega(n^{1+1/k})$ edges. This conjecture has been proved for $k = 1, 2, 3$ and 5. Note that a $(2k-1)$-spanner is also a $2k$-spanner and the lower bound on the size is the same for both a $2k$-spanner and a $(2k-1)$-spanner. So the objective is to design an algorithm that, for any weighted graph on n vertices, computes a $(2k-1)$-spanner of $O(n^{1+1/k})$ size. Needless to say, one would like to design the fastest algorithm for this problem, and the most ambitious aim would be to achieve the linear time complexity.

Key Results

The key results of this article are two very simple algorithms which compute a $(2k-1)$-spanner of a given weighted graph $G = (V, E)$. Let n and m denote the number of vertices and edges of G, respectively. The first algorithm, due to Althöfer et al. [2], is based on a greedy strategy, and runs in $O(mn^{1+1/k})$ time. The second algorithm [6] is based on a very local approach and runs in the expected $O(km)$ time. To start with, consider the following simple observation. Suppose there is a subset $E_S \subset E$ that ensures the following proposition for every edge $(x, y) \in E \backslash E_S$.

$\mathcal{P}_t(x, y)$: the vertices x and y are connected in the subgraph (V, E_S) by a path consisting of at most t edges, and the weight of each edge on this path is not more than that of the edge (x, y).

It follows easily that the sub graph (V, E_S) will be a t-spanner of G. The two algorithms for computing the $(2k-1)$-

spanner eventually compute the set E_S based on two completely different approaches.

Algorithm I

This algorithm selects edges for its spanner in a greedy fashion, and is similar to Kruskal's algorithm for computing a minimum spanning tree. The edges of the graph are processed in the increasing order of their weights. To begin with, the spanner $E_S = \emptyset$ and the algorithm adds edges to it gradually. The decision as to whether an edge, say (u, v), has to be added (or not) to E_S is made as follows:

If the distance between u and v in the subgraph induced by the current spanner edges E_S is more than $t\cdot$weight(u, v), then add the edge (u, v) to E_S, otherwise discard the edge.

It follows that $\mathcal{P}_t(x, y)$ would hold for each edge of E missing in E_S, and so at the end, the subgraph (V, E_S) will be a t-spanner. A well known result in elementary graph theory states that a graph with more than $n^{1+1/k}$ edges must have a cycle of length at most $2k$. It follows from the above algorithm that the length of any cycle in the subgraph (V, E_S) has to be at least $t + 1$. Hence for $t = 2k - 1$, the number of edges in the subgraph (V, E_S) will be less than $n^{1+1/k}$. Thus Algorithm I computes a $(2k - 1)$-spanner of size $O(n^{1+1/k})$, which is indeed optimal based on the lower bound mentioned earlier.

A simple $O(mn^{1+1/k})$ implementation of Algorithm I follows based on Dijkstra's algorithm. Cohen [9], and later Thorup and Zwick [18] designed algorithms for a $(2k - 1)$-spanner with an improved running time of $O(kmn^{1/k})$. These algorithms rely on several calls to Dijkstra's single-source shortest-path algorithm for distance computation and therefore were far from achieving linear time. On the other hand, since a spanner must approximate all pairs distances in a graph, it appears difficult to compute a spanner by avoiding explicit distance information. Somewhat surprisingly, Algorithm II, described in the following section, avoids any sort of distance computation and achieves expected linear time.

Algorithm II

This algorithm employs a novel clustering based on a very local approach, and establishes the following result for the spanner problem.

> Given a weighted graph $G = (V, E)$, and an integer $k > 1$, a spanner of $(2k - 1)$-stretch and $O(kn^{1+1/k})$ size can be computed in expected $O(km)$ time.

The algorithm executes in $O(k)$ rounds, and in each round it essentially explores adjacency list of each vertex to prune dispensable edges. As a testimony of its simplicity, we will present the entire algorithm for a 3-spanner and its analysis in the following section. The algorithm can be easily adapted in other computational models (parallel, external memory, distributed) with nearly optimal performance (see [6] for more details).

Computing a 3-Spanner in Linear Time To meet the size constraint of a 3-spanner, a vertex should contribute an average of $\sqrt{n}$ edges to the spanner. So the vertices with degree $O(\sqrt{n})$ are easy to handle since all their edges can be selected in the spanner. For vertices with higher degree a clustering (grouping) scheme is employed to tackle this problem which has its basis in *dominating sets*.

To begin with, there is a set of edges E' initialized to E, and an empty spanner E_S. The algorithm processes the edges E', moves some of them to the spanner E_S and discards the remaining ones. It does so in the following two phases.

1. *Forming the clusters*:
   A sample $\mathcal{R} \subset V$ is chosen by picking each vertex independently with probability $1/\sqrt{n}$. The clusters will be formed around these sampled vertices. Initially the clusters are $\{\{u\} | u \in \mathcal{R}\}$. Each $u \in \mathcal{R}$ is called the *center* of its cluster. Each unsampled vertex $v \in V - \mathcal{R}$ is processed as follows.
   (a) If v is not adjacent to any sampled vertex, then every edge incident on v is moved to E_S.
   (b) If v is adjacent to one or more sampled vertices, let $\mathcal{N}(v, \mathcal{R})$ be the sampled neighbor that is nearest[1] to v. The edge $(v, \mathcal{N}(v, \mathcal{R}))$ along with every edge that is incident on v with weight less than this edge is moved to E_S. The vertex v is added to the cluster centered at $\mathcal{N}(v, \mathcal{R})$.

   As a last step of the first phase, all those edges (u, v) from E' where u and v are not sampled and belong to the same cluster are discarded.
   Let V' be the set of vertices corresponding to the endpoints of the edges E' left after the first phase. It follows that each vertex from V' is either a sampled vertex or adjacent to some sampled vertex, and the step 1(b) has partitioned V' into disjoint clusters, each centered around some sampled vertex. Also note that, as a consequence of the last step, each edge of the set E' is an inter-cluster edge. The graph (V', E'), and the corresponding clustering of V' is passed onto the following (second) phase.
2. *Joining vertices with their neighboring clusters*:
   Each vertex v of graph (V', E') is processed as follows.

[1]Ties can be broken arbitrarily. However, it helps conceptually to assume that all weights are distinct.

Let $E'(v, c)$ be the edges from the set E' incident on v from a cluster c. For each cluster c that is a neighbor of v, the least-weight edge from $E'(v, c)$ is moved to E_S and the remaining edges are discarded.

The number of edges added to the spanner E_S during the algorithm described above can be bounded as follows. Note that the sample set $\mathcal{R}$ is formed by picking each vertex randomly and independently with probability $1/\sqrt{n}$. It thus follows from elementary probability that for each vertex $v \in V$, the expected number of incident edges with weights less than that of $(v, \mathcal{N}(v, \mathcal{R}))$ is at most $\sqrt{n}$. Thus the expected number of edges contributed to the spanner by each vertex in the first phase of the algorithm is at most $\sqrt{n}$. The number of edges added to the spanner in the second phase is $O(n|\mathcal{R}|)$. Since the expected size of the sample $\mathcal{R}$ is $\sqrt{n}$, therefore, the expected number of edges added to the spanner in the second phase is at most $n^{3/2}$. Hence the expected size of the spanner E_S at the end of Algorithm II as described above is at most $2n^{3/2}$. The algorithm is repeated if the size of the spanner exceeds $3n^{3/2}$. It follows using Markov's inequality that the expected number of such repetitions will be $O(1)$.

We now establish that E_S is a 3-spanner. Note that for every edge $(u, v) \notin E_S$, the vertices u and v belong to some cluster in the first phase. There are two cases now.

Case 1 : *(u and v belong to same cluster)*
Let u and v belong to the cluster centered at $x \in \mathcal{R}$. It follows from the first phase of the algorithm that there is a 2-edge path $u - x - v$ in the spanner with each edge not heavier than the edge (u, v). (This provides a justification for discarding all intra-cluster edges at the end of first phase).

Case 2 : *(u and v belong to different clusters)*
Clearly the edge (u, v) was removed from E' during phase 2, and suppose it was removed while processing the vertex u. Let v belong to the cluster centered at $x \in \mathcal{R}$.

In the beginning of the second phase let $(u, v') \in E'$ be the least weight edge among all the edges incident on u from the vertices of the cluster centered at x. So it must be that weight$(u, v') \leq$ weight(u, v). The processing of vertex u during the second phase of our algorithm ensures that the edge (u, v') gets added to E_S. Hence there is a path $\Pi_{uv} = u - v' - x - v$ between u and v in the spanner E_S, and its weight can be bounded as weight(Π_{uv}) = weight(u, v') + weight(v', x) + weight(x, v). Since (v', x) and (v, x) were chosen in the first phase, it follows that weight$(v', x) \leq$ weight(u, v') and weight$(x, v) \leq$ weight(u, v). It follows that the spanner (V, E_S) has stretch 3. Moreover, both phases of the algorithm can be executed in $O(m)$ time using elementary data structures and bucket sorting.

The algorithm for computing a $(2k - 1)$-spanner executes k iterations where each iteration is similar to the first phase of the 3-spanner algorithm. For details and formal proofs, the reader may refer to [6].

Other Related Work

The notion of a spanner has been generalized in the past by many researchers.

Additive spanners: A t-spanner as defined above approximates pairwise distances with multiplicative error, and can be called a multiplicative spanner. In an analogous manner, one can define spanners that approximate pairwise distances with additive error. Such a spanner is called an additive spanner and the corresponding error is called a *surplus*. Aingworth et al. [1] presented the first additive spanner of size $O(n^{3/2} \log n)$ with surplus 2. Baswana et al. [7] presented a construction of $O(n^{4/3})$ size additive spanner with surplus 6. It is a major open problem if there exists any sparser additive spanner.

(α, β)-spanner: Elkin and Peleg [11] introduced the notion of an (α, β)-spanner for unweighted graphs, which can be viewed as a hybrid of multiplicative and additive spanners. An (α, β)-spanner is a subgraph such that the distance between any pair of vertices $u, v \in V$ in this subgraph is bounded by $\alpha\delta(u, v) + \beta$, where $\delta(u, v)$ is the distance between u and v in the original graph. Elkin and Peleg showed that an $(1 + \epsilon, \beta)$-spanner of size $O(\beta n^{1+\delta})$, for arbitrarily small $\epsilon, \delta > 0$, can be computed at the expense of a sufficiently large surplus β. Recently Thorup and Zwick [19] introduced a spanner where the additive error is sublinear in terms of the *distance* being approximated.

Other interesting variants of spanners include the *distance preserver* proposed by Bollobás et al. [8] and the *Light-weight* spanner proposed by Awerbuch et al. [4]. A subgraph is said to be a d-preserver if it preserves exact distances for each pair of vertices which are separated by distance at least d. A light-weight spanner tries to minimize the number of edges as well as the total edge weight. A *lightness* parameter is defined for a subgraph as the ratio of the total weight of all its edges and the weight of the minimum spanning tree of the graph. Awerbuch et al. [4] showed that for any weighted graph and integer $k > 1$, there exists a polynomially constructable $O(k)$-spanner with $O(k\rho n^{1+1/k})$ edges and $O(k\rho n^{1/k})$ lightness, where $\rho = \log(\text{Diameter})$.

In addition to the above work on the generalization of spanners, a lot of work has also been done on computing spanners for special classes of graphs, e. g., chordal graphs, unweighted graphs, and Euclidean graphs. For chordal graphs, Peleg and Schäffer [14] designed an algorithm that computes a 2-spanner of size $O(n^{3/2})$, and a 3-spanner of size $O(n \log n)$. For unweighted graphs Halperin and Zwick [13] gave an $O(m)$ time algorithm for this problem. Salowe [17] presented an algorithm for computing a $(1+\epsilon)$-spanner of a d-dimensional complete Euclidean graph in $O(n \log n + \frac{n}{\epsilon^d})$ time. However, none of the algorithms for these special classes of graphs seem to extend to general weighted undirected graphs.

Applications

Spanners are quite useful in various applications in the areas of distributed systems and communication networks. In these applications, spanners appear as the underlying graph structure. In order to build compact routing tables [16], many existing routing schemes use the edges of a sparse spanner for routing messages. In distributed systems, spanners play an important role in designing *synchronizers*. Awerbuch [3], and Peleg and Ullman [15] showed that the quality of a spanner (in terms of stretch factor and the number of spanner edges) is very closely related to the time and communication complexity of any synchronizer for the network. The spanners have also been used implicitly in a number of algorithms for computing all pairs of approximate shortest paths [5,9,18]. For a number of other applications, please refer to the papers [2,3,14,16].

Open Problems

The running time as well as the size of the $(2k-1)$-spanner computed by the Algorithm II described above are away from their respective worst case lower bounds by a factor of k. For any constant value of k, both these parameters are optimal. However, for the extreme value of k, that is, for $k = \log n$, there is a deviation by a factor of $\log n$. Is it possible to get rid of this multiplicative factor of k from the running time of the algorithm and/or the size of the $(2k-1)$-spanner computed? It seems that a more careful analysis coupled with advanced probabilistic tools might be useful in this direction.

Recommended Reading

1. Aingworth, D., Chekuri, C., Indyk, P., Motwani, R.: Fast estimation of diameter and shortest paths (without matrix multiplication). SIAM J. Comput. **28**, 1167–1181 (1999)
2. Althöfer, I., Das, G., Dobkin, D.P., Joseph, D., Soares J.: On sparse spanners of weighted graphs. Discret. Comput. Geom. **9**, 81–100 (1993)
3. Awerbuch, B.: Complexity of network synchronization. J. Assoc. Comput. Mach. **32**(4), 804–823 (1985)
4. Awerbuch, B., Baratz, A., Peleg, D.: Efficient broadcast and light weight spanners. Tech. Report CS92-22, Weizmann Institute of Science (1992)
5. Awerbuch, B., Berger, B., Cowen, L., Peleg D.: Near-linear time construction of sparse neighborhood covers. SIAM J. Comput. **28**, 263–277 (1998)
6. Baswana, S., Sen, S.: A simple and linear time randomized algorithm for computing sparse spanners in weighted graphs. Random Struct. Algorithms **30**, 532–563 (2007)
7. Baswana, S., Telikepalli, K., Mehlhorn, K., Pettie, S.: New construction of (α, β)-spanners and purely additive spanners. In: Proceedings of 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 2005, pp. 672–681
8. Bollobás, B., Coppersmith, D., Elkin M.: Sparse distance preserves and additive spanners. In: Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 2003, pp. 414–423
9. Cohen, E.: Fast algorithms for constructing t-spanners and paths with stretch t. SIAM J. Comput. **28**, 210–236 (1998)
10. Elkin, M., Peleg, D.: Strong inapproximability of the basic k-spanner problem. In: Proc. of 27th International Colloquim on Automata, Languages and Programming, 2000, pp. 636–648
11. Elkin, M., Peleg, D.: $(1+\epsilon, \beta)$-spanner construction for general graphs. SIAM J. Comput. **33**, 608–631 (2004)
12. Erdös, P.: Extremal problems in graph theory. In: Theory of Graphs and its Applications (Proc. Sympos. Smolenice, 1963), pp. 29–36. Publ. House Czechoslovak Acad. Sci., Prague (1964)
13. Halperin, S., Zwick, U.: Linear time deterministic algorithm for computing spanners for unweighted graphs. unpublished manuscript (1996)
14. Peleg, D., Schäffer, A.A.: Graph spanners. J. Graph Theory **13**, 99–116 (1989)
15. Peleg, D., Ullman, J.D.: An optimal synchronizer for the hypercube. SIAM J. Comput. **18**, 740–747 (1989)
16. Peleg, D., Upfal, E.: A trade-off between space and efficiency for routing tables. J. Assoc. Comput Mach. **36**(3), 510–530 (1989)
17. Salowe, J.D.: Construction of multidimensional spanner graphs, with application to minimum spanning trees. In: ACM Symposium on Computational Geometry, 1991, pp. 256–261
18. Thorup, M., Zwick, U.: Approximate distance oracles. J. Assoc. Comput. Mach. **52**, 1–24 (2005)
19. Thorup, M., Zwick, U.: Spanners and emulators with sublinear distance errors. In: Proceedings of 17th Annual ACM-SIAM Symposium on Discrete Algorithms, 2006, pp. 802–809

All Pairs Shortest Paths in Sparse Graphs

2004; Pettie

SETH PETTIE
Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI, USA

Keywords and Synonyms

Shortest route; Quickest route

Problem Definition

Given a communications network or road network one of the most natural algorithmic questions is how to determine the shortest path from one point to another. The *all pairs* shortest path problem (APSP) is, given a directed graph $G = (V, E, \ell)$, to determine the distance and shortest path between every pair of vertices, where $|V| = n$, $|E| = m$, and $\ell\colon E \to \mathbb{R}$ is the edge length (or weight) function. The output is in the form of two $n \times n$ matrices: $D(u, v)$ is the distance from u to v and $S(u, v) = w$ if (u, w) is the first edge on a shortest path from u to v. The APSP problem is often contrasted with the *point-to-point* and *single source* (SSSP) shortest path problems. They ask for, respectively, the shortest path from a given source vertex to a given target vertex, and all shortest paths from a given source vertex.

Definition of Distance

If ℓ assigns only non-negative edge lengths then the definition of distance is clear: $D(u, v)$ is the length of the minimum length path from u to v, where the length of a path is the total length of its constituent edges. However, if ℓ can assign negative lengths then there are several sensible notations of distance that depend on how negative length cycles are handled. Suppose that a cycle C has negative length and that $u, v \in V$ are such that C is reachable from u and v reachable from C. Because C can be traversed an arbitrary number of times when traveling from u to v, there is no shortest path from u to v using a finite number of edges. It is sometimes assumed a priori that G has no negative length cycles; however it is cleaner to define $D(u, v) = -\infty$ if there is no finite shortest path. If $D(u, v)$ is defined to be the length of the shortest *simple* path (no repetition of vertices) then the problem becomes NP-hard.[1] One could also define distance to be the length of the shortest path without repetition of edges.

Classic Algorithms

The Bellman-Ford algorithm solves SSSP in $O(mn)$ time and under the assumption that edge lengths are non-negative, Dijkstra's algorithm solves it in $O(m + n \log n)$ time. There is a well known $O(mn)$-time shortest path preserving transformation that replaces any length function with a non-negative length function. Using this transformation and n runs of Dijkstra's algorithm gives an APSP algorithm running in $O(mn + n^2 \log n) = O(n^3)$ time. The Floyd–Warshall algorithm computes APSP in a more direct manner, in $O(n^3)$ time. Refer to [4] for a description of these algorithms. It is known that APSP on complete graphs is asymptotically equivalent to $(\min, +)$ matrix multiplication [1], which can be computed by a non-uniform algorithm that performs $O(n^{2.5})$ numerical operations [6].[2]

Integer-Weighted Graphs

Much recent work on shortest paths assume that edge lengths are integers in the range $\{-C, \dots, C\}$ or $\{0, \dots, C\}$. One line of research reduces APSP to a series of standard matrix multiplications. These algorithms are limited in their applicability because their running times scale linearly with C. There are faster SSSP algorithms for both non-negative edge lengths and arbitrary edge lengths. The former exploit the power of RAMs to sort in $o(n \log n)$ time and the latter are based on the *scaling* technique. See Zwick [19] for a survey of shortest path algorithms up to 2001.

Key Results

Pettie's APSP algorithm [13] adapts the *hierarchy* approach of Thorup [17] (designed for undirected, integer-weighted graphs) to general real-weighted directed graphs. Theorem 1 is the first improvement over the $O(mn + n^2 \log n)$ time bound of Dijkstra's algorithm on arbitrary real-weighted graphs.

Theorem 1 *Given a real-weighted directed graph, all pairs shortest paths can be solved in* $O(mn + n^2 \log\log n)$ *time.*

This algorithm achieves a logarithmic speedup through a trio of new techniques. The first is to exploit the necessary similarity between the SSSP trees emanating from nearby vertices. The second is a method for computing discrete approximate distances in real-weighted graphs. The third is a new hierarchy-type SSSP algorithm that runs in $O(m + n \log\log n)$ time when given suitably accurate approximate distances.
Theorem 1 should be contrasted with the time bounds of other hierarchy-type APSP algorithms [17,12,15].

Theorem 2 ([15], 2005) *Given a real-weighted undirected graph, APSP can be solved in* $O(mn \log \alpha(m, n))$ *time.*

Theorem 3 ([17], 1999) *Given an undirected graph* $G(V, E, \ell)$*, where* ℓ *assigns integer edge lengths in the range* $\{-2^{w-1}, \dots, 2^{w-1} - 1\}$*, APSP can be solved in* $O(mn)$ *time on a RAM with* w*-bit word length.*

[1]If all edges have length -1 then $D(u, v) = -(n - 1)$ if and only if G contains a Hamiltonian path [7] from u to v.

[2]The fastest known $(\min, +)$ matrix multiplier runs n $O(n^3(\log\log n)^3/(\log n)^2)$ time [3].

Theorem 4 ([14], 2002) *Given a real-weighted directed graph, APSP can be solved in polynomial time by an algorithm that performs $O(mn \log \alpha(m, n))$ numerical operations, where α is the inverse-Ackermann function.*

A secondary result of [13,15] is that no *hierarchy-type* shortest path algorithm can improve on the $O(m + n \log n)$ running time of Dijkstra's algorithm.

Theorem 5 *Let G be an input graph such that the ratio of the maximum to minimum edge length is r. Any hierarchy-type SSSP algorithm performs $\Omega(m + \min\{n \log n, n \log r\})$ numerical operations if G is directed and $\Omega(m + \min\{n \log n, n \log\log r\})$ if G is undirected.*

Applications

Shortest paths appear as a subproblem in other graph optimization problems; the minimum weight perfect matching, minimum cost flow, and minimum mean-cycle problems are some examples. A well known commercial application of shortest path algorithms is finding efficient routes on road networks; see, for example, Google Maps, MapQuest, or Yahoo Maps.

Open Problems

The longest standing open shortest path problems are to improve the SSSP algorithms of Dijkstra's and Bellman-Ford on *real*-weighted graphs.

Problem 1 *Is there an $o(mn)$ time SSSP or point-to-point shortest path algorithm for arbitrarily weighted graphs?*

Problem 2 *Is there an $O(m) + o(n \log n)$ time SSSP algorithm for directed, non-negatively weighted graphs? For undirected graphs?*

A partial answer to Problem 2 appears in [15], which considers undirected graphs. Perhaps the most surprising open problem is whether there is any (asymptotic) difference between the complexities of the all pairs, single source, and point-to-point shortest path problems on arbitrarily weighted graphs.

Problem 3 *Is point-to-point shortest paths easier than all pairs shortest paths on arbitrarily weighted graphs?*

Problem 4 *Is there a genuinely subcubic APSP algorithm, i. e., one running in time $O(n^{3-\epsilon})$? Is there a subcubic APSP algorithm for integer-weighted graphs with weak dependence on the largest edge weight C, i. e., running in time $O(n^{3-\epsilon}\operatorname{polylog}(C))$?*

Experimental Results

See [9,16,5] for recent experiments on SSSP algorithms. On sparse graphs the best APSP algorithms use repeated application of an SSSP algorithm, possibly with some precomputation [16]. On dense graphs cache-efficiency becomes a major issue. See [18] for a cache conscious implementation of the Floyd–Warshall algorithm.

The trend in recent years is to construct a linear space data structure that can quickly answer exact or approximate point-to-point shortest path queries; see [10,6,2,11].

Data Sets

See [5] for a number of U.S. and European road networks.

URL to Code

See [8] and [5].

Cross References

► All Pairs Shortest Paths via Matrix Multiplication
► Single-Source Shortest Paths

Recommended Reading

1. Aho, A.V., Hopcroft, J.E., Ullman, J.D.: The design and analysis of computer algorithms. Addison-Wesley, Reading (1975)
2. Bast, H., Funke, S., Matijevic, D., Sanders, P., Schultes, D.: In transit to constant shortest-path queries in road networks. In: Proc. 9th Workshop on Algorithm Engineering and Experiments (ALENEX), 2007
3. Chan, T.: More algorithms for all-pairs shortest paths in weighted graphs. In: Proc. 39th ACM Symposium on Theory of Computing (STOC), 2007, pp. 590–598
4. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms. MIT Press, Cambridge (2001)
5. Demetrescu, C., Goldberg, A.V., Johnson, D.: 9th DIMACS Implementation challenge – shortest paths. http://www.dis.uniroma1.it/~challenge9/ (2006)
6. Fredman, M.L.: New bounds on the complexity of the shortest path problem. SIAM J. Comput. **5**(1), 83–89 (1976)
7. Garey, M.R., Johnson, D.S.: Computers and Intractability: a guide to NP-Completeness. Freeman, San Francisco (1979)
8. Goldberg, A.V.: AVG Lab. http://www.avglab.com/andrew/
9. Goldberg, A.V.: Shortest path algorithms: Engineering aspects. In: Proc. 12th Int'l Symp. on Algorithms and Computation (ISAAC). LNCS, vol. 2223, pp. 502–513. Springer, Berlin (2001)
10. Goldberg, A.V., Kaplan, H., Werneck, R.: Reach for A*: efficient point-to-point shortest path algorithms. In: Proc. 8th Workshop on Algorithm Engineering and Experiments (ALENEX), 2006
11. Knopp, S., Sanders, P., Schultes, D., Schulz, F., Wagner, D.: Computing many-to-many shortest paths using highway hierarchies. In: Proc. 9th Workshop on Algorithm Engineering and Experiments (ALENEX), 2007

12. Pettie, S.: On the comparison-addition complexity of all-pairs shortest paths. In: Proc. 13th Int'l Symp. on Algorithms and Computation (ISAAC), 2002, pp. 32–43
13. Pettie, S.: A new approach to all-pairs shortest paths on real-weighted graphs. Theor. Comput. Sci. **312**(1), 47–74 (2004)
14. Pettie, S., Ramachandran, V.: Minimizing randomness in minimum spanning tree, parallel connectivity and set maxima algorithms. In: Proc. 13th ACM-SIAM Symp. on Discrete Algorithms (SODA), 2002, pp. 713–722
15. Pettie, S., Ramachandran, V.: A shortest path algorithm for real-weighted undirected graphs. SIAM J. Comput. **34**(6), 1398–1431 (2005)
16. Pettie, S., Ramachandran, V., Sridhar, S.: Experimental evaluation of a new shortest path algorithm. In: Proc. 4th Workshop on Algorithm Engineering and Experiments (ALENEX), 2002, pp. 126–142
17. Thorup, M.: Undirected single-source shortest paths with positive integer weights in linear time. J. ACM **46**(3), 362–394 (1999)
18. Venkataraman, G., Sahni, S., Mukhopadhyaya, S.: A blocked all-pairs shortest paths algorithm. J. Exp. Algorithms **8** (2003)
19. Zwick, U.: Exact and approximate distances in graphs – a survey. In: Proc. 9th European Symposium on Algorithms (ESA), 2001, pp. 33–48. See updated version at http://www.cs.tau.ac.il/~zwick/

All Pairs Shortest Paths via Matrix Multiplication

2002; Zwick

TADAO TAKAOKA
Department of Computer Science and Software Engineering, University of Canterbury, Christchurch, New Zealand

Keywords and Synonyms

Shortest path problem; Algorithm analysis

Problem Definition

The all pairs shortest path (APSP) problem is to compute shortest paths between all pairs of vertices of a directed graph with non-negative real numbers as edge costs. Focus is given on shortest distances between vertices, as shortest paths can be obtained with a slight increase of cost. Classically, the APSP problem can be solved in cubic time of $O(n^3)$. The problem here is to achieve a sub-cubic time for a graph with small integer costs.

A directed graph is given by $G = (V, E)$, where $V = \{1, \dots, n\}$, the set of vertices, and E is the set of edges. The cost of edge $(i, j) \in E$ is denoted by d_{ij}. The (n, n)-matrix D is one whose (i,j) element is d_{ij}. It is assumed for simplicity that $d_{ij} > 0$ and $d_{ii} = 0$ for all $i \neq j$. If there is no edge from i to j, let $d_{ij} = \infty$. The cost, or distance, of a path is the sum of costs of the edges in the path. The length of a path is the number of edges in the path. The shortest distance from vertex i to vertex j is the minimum cost over all paths from i to j, denoted by d^*_{ij}. Let $D^* = \{d^*_{ij}\}$. The value of n is called the size of the matrices.

Let A and B be (n, n)-matrices. The three products are defined using the elements of A and B as follows: (1) Ordinary matrix product over a ring $C = AB$, (2) Boolean matrix product $C = A \cdot B$, and (3) Distance matrix product $C = A \times B$, where

$$(1)\ c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}\,, \quad (2)\ c_{ij} = \bigvee_{k=1}^{n} a_{ik} \wedge b_{kj}\,,$$

$$(3)\ c_{ij} = \min_{1 \le k \le n} \{a_{ik} + b_{kj}\}\,.$$

The matrix C is called a product in each case; the computational process is called multiplication, such as distance matrix multiplication. In those three cases, k changes through the entire set $\{1, \dots, n\}$. A partial matrix product of A and B is defined by taking k in a subset I of V. In other words, a partial product is obtained by multiplying a vertically rectangular matrix, $A(*, I)$, whose columns are extracted from A corresponding to the set I, and similarly a horizontally rectangular matrix, $B(I, *)$, extracted from B with rows corresponding to I. Intuitively I is the set of check points k, when going from i to j in the graph.

The best algorithm [3] computes (1) in $O(n^\omega)$ time, where $\omega = 2.376$. Three decimal points are carried throughout this article. To compute (2), Boolean values 0 and 1 in A and B can be regarded as integers and use the algorithm for (1), and convert non-zero elements in the resulting matrix to 1. Therefore, this complexity is $O(n^\omega)$. The witnesses of (2) are given in the witness matrix $W = \{w_{ij}\}$ where $w_{ij} = k$ for some k such that $a_{ik} \wedge b_{kj} = 1$. If there is no such k, $w_{ij} = 0$. The witness matrix $W = \{w_{ij}\}$ for (3) is defined by $w_{ij} = k$ that gives the minimum to c_{ij}. If there is an algorithm for (3) with $T(n)$ time, ignoring a polylog factor of n, the APSP problem can be solved in $\tilde{O}(T(n))$ time by the repeated squaring method, described as the repeated use of $D \leftarrow D \times D$ $O(\log n)$ times.

The definition here of computing shortest paths is to give a witness matrix of size n by which a shortest path from i to j can be given in $O(\ell)$ time where ℓ is the length of the path. More specifically, if $w_{ij} = k$ in the witness matrix $W = \{w_{ij}\}$, it means that the path from i to j goes through k. Therefore, a recursive function $path(i,j)$ is defined by

$(path(i,k), k, path(k,j))$ if $w_{ij} = k > 0$ and nil if $w_{ij} = 0$, where a path is defined by a list of vertices excluding endpoints. In the following sections, k is recorded in w_{ij} whenever k is found such that a path from i to j is modified or newly set up by paths from i to k and from k to j. Preceding results are introduced as a framework for the key results.

Alon–Galil–Margalit Algorithm

The algorithm by Alon, Galil, and Margalit [1] is reviewed. Let the costs of edges of the given graph be ones. Let $D^{(\ell)}$ be the ℓth approximate matrix for D^* defined by $d_{ij}^{(\ell)} = d_{ij}^*$ if $d_{ij}^* \le \ell$, and $d_{ij}^{(\ell)} = \infty$ otherwise. Let A be the adjacency matrix of G, that is, $a_{ij} = 1$ if there is an edge (i,j), and $a_{ij} = 0$ otherwise. Let $a_{ii} = 1$ for all i. The algorithm consists of two phases. In the first phase, $D^{(\ell)}$ is computed for $\ell = 1, \ldots, r$, by checking the (i,j)-element of $A^\ell = \{a_{ij}^\ell\}$. Note that if $a_{ij}^\ell = 1$, there is a path from i to j of length ℓ or less. Since Boolean mutrix multiplication can be computed in $O(n^\omega)$ time, the computing time of this part is $O(rn^\omega)$.

In the second phase, the algorithm computes $D^{(\ell)}$ for $\ell = r, \lceil 3/2\, r \rceil, \lceil 3/2 \lceil 3/2\, r \rceil \rceil, \cdots, n'$ by repeated squaring, where n' is the smallest integer in this sequence of ℓ such that $\ell \ge n$. Let $T_{i\alpha} = \{j | d_{ij}^{(\ell)} = \alpha\}$, and $I_i = T_{i\alpha}$ such that $|T_{i\alpha}|$ is minimum for $\lceil \ell/2 \rceil \le \alpha \le \ell$. The key observation in the second phase is that it is only needed to check k in I_i whose size is not larger than $2n/\ell$, since the correct distances between $\ell + 1$ and $\lceil 3\ell/2 \rceil$ can be obtained as the sum $d_{ik}^{(\ell)} + d_{kj}^{(\ell)}$ for some k satisfying $\lceil \ell/2 \rceil \le d_{ik}^{(\ell)} \le \ell$. The meaning of I_i is similar to I for partial products except that I varies for each i. Hence the computing time of one squaring is $O(n^3/\ell)$. Thus, the time of the second phase is given with $N = \lceil \log_{3/2} n/r \rceil$ by $O\left(\sum_{s=1}^{N} n^3/((3/2)^s r)\right) = O(n^3/r)$. Balancing the two phases with $rn^\omega = n^3/r$ yields $O(n^{(\omega+3)/2}) = O(n^{2.688})$ time for the algorithm with $r = O(n^{(3-\omega)/2})$.

Witnesses can be kept in the first phase in time polylog of n by the method in [2]. The maintenance of witnesses in the second phase is straightforward.

When a directed graph G whose edge costs are integers between 1 and M is given, where M is a positive integer, the graph G can be converted to G' by replacing each edge by up to M edges with unit cost. Obviously the problem for G can be solved by applying the above algorithm to G', which takes $O((Mn)^{(\omega+3)/2})$ time. This time is sub-cubic when $M < n^{0.116}$. The maintenance of witnesses has an extra polylog factor in each case.

For undirected graphs with unit edge costs, $\tilde{O}(n^\omega)$ time is known in Seidel [7].

Takaoka algorithm

When the edge costs are bounded by a positive integer M, a better algorithm can be designed than in the above as shown in Takaoka [9]. Romani's algorithm [6] for distance matrix multiplication is reviewed briefly.

Let A and B be (n,m) and (m,n) distance matrices whose elements are bounded by M or infinite. Let the diagonal elements be 0. A and B are converted into A' and B' where $a'_{ij} = (m+1)^{M-a_{ij}}$, if $a_{ij} \ne \infty$, 0 otherwise, and $b'_{ij} = (m+1)^{M-b_{ij}}$, if $b_{ij} \ne \infty$, 0 otherwise.

Let $C' = A'B'$ be the product by ordinary matrix multiplication and $C = A \times B$ be that by distance matrix multiplication. Then it holds that

$$c'_{ij} = \sum_{k=1}^{m} (m+1)^{2M-(a_{ik}+b_{kj})}, \quad c_{ij} = 2M - \lfloor \log_{m+1} c'_{ij} \rfloor .$$

This distance mutrix multiplication is called (n,m)-Romani. In this section the above multiplication is used with square matrices, that is, (n,n)-Romani is used. In the next section, the case where $m < n$ is dealt with.

C can be computed with $O(n^\omega)$ arithmetic operations on integers up to $(n+1)^M$. Since these values can be expressed by $O(M \log n)$ bits and Schönhage and Strassen's algorithm [8] for multiplying k-bit numbers takes $O(k \log k \log\log k)$ bit operations, C can be computed in $O(n^\omega M \log n \log(M \log n) \log\log(M \log n))$ time, or $\tilde{O}(Mn^\omega)$ time.

The first phase is replaced by the one based on (n,n)-Romani, and modify the second phase based on path lengths, not distances.

Note that the bound M is replaced by ℓM in the distance matrix multiplication in the first phase. Ignoring polylog factors, the time for the first phase is given by $\tilde{O}(n^\omega r^2 M)$. It is assumed that M is $O(n^k)$ for some constant k. Balancing this complexity with that of the second phase, $O(n^3/r)$, yields the total computing time of $\tilde{O}(n^{(6+\omega)/3} M^{1/3})$ with the choice of $r = O(n^{(3-\omega)/3} M^{-1/3})$. The value of M can be almost $O(n^{0.624})$ to keep the complexity within sub-cubic.

Key Results

Zwick improved the Alon–Galil–Margalit algorithm in several ways. The most notable is an improvement of the time for the APSP problem with unit edge costs from $O(n^{2.688})$ to $O(n^{2.575})$. The main accelerating engine in Alon–Galil–Margalit [1] was the fast Boolean matrix multiplication and that in Takaoka [9] was the fast distance matrix multiplication by Romani, both powered by the fast matrix multiplication of square matrices.

In this section, the engine is the fast distance matrix multiplication by Romani powered by the fast matrix multiplication of rectangular matrices given by Coppersmith [4], and Huang and Pan [5]. Let $\omega(p, q, r)$ be the exponent of time complexity for multiplying (n^p, n^q) and (n^q, n^r) matrices. Suppose the product of (n, m) matrix and (m, n) matrix can be computed with $O(n^{\omega(1,\mu,1)})$ arithmetic operations, where $m = n^\mu$ with $0 \le \mu \le 1$. Several facts such as $O(n^{\omega(1,1,1)}) = O(n^{2.376})$ and $O(n^{\omega(1,0.294,1)}) = \tilde{O}(n^2)$ are known. To compute the product of (n, n) square matrices, $n^{1-\mu}$ matrix multiplications are needed, resulting in $O(n^{\omega(1,\mu,1)+1-\mu})$ time, which is reformulated as $O(n^{2+\mu})$, where μ satisfies the equation $\omega(1, \mu, 1) = 2\mu + 1$. Currently the best-known value for μ is $\mu = 0.575$, so the time becomes $O(n^{2.575})$, which is not as good as $O(n^{2.376})$. So the algorithm for rectangular matrices is used in the following.

The above algorithm is incorporated into (n, m)-Romani with $m = n^\mu$ and $M = n^t$ for some $t > 0$, and the computing time of $\tilde{O}(Mn^{\omega(1,\mu,1)})$. The next step is how to incorporate (n, m)-Romani into the APSP algorithm. The first algorithm is a mono-phase algorithm based on repeated squaring, similar to the second phase of the algorithm in [1]. To take advantage of rectangular matrices in (n, m)-Romani, the following definition of the bridging set is needed, which plays the role of the set I in the partial distance matrix product in Sect. "Problem Definition".

Let $\delta(i, j)$ be the shortest distance from i to j, and $\eta(i, j)$ be the minimum length of all shortest paths from i to j. A subset I of V is an ℓ-bridging set if it satisfies the condition that if $\eta(i, j) \ge \ell$, there exists $k \in I$ such that $\delta(i, j) = \delta(i, k) + \delta(k, j)$. I is a strong ℓ-bridging set if it satisfies the condition that if $\eta(i, j) \ge \ell$, there exists $k \in I$ such that $\delta(i, j) = \delta(i, k) + \delta(k, j)$ and $\eta(i, j) = \eta(i, k) + \eta(k, j)$. Note that those two sets are the same for a graph with unit edge costs.

Note that if $(2/3)\ell \le \mu(i, j) \le \ell$ and I is a strong $\ell/3$-bridging set, there is a $k \in I$ such that $\delta(i, j) = \delta(i, k) + \delta(k, j)$ and $\mu(i, j) = \mu(i, k) + \mu(k, j)$. With this property of strong bridging sets, (n, m)-Romani can be used for the APSP problem in the following way. By repeated squaring in a similar way to Alon–Galil–Margalit, the algorithm computes $D^{(\ell)}$ for $\ell = 1, \lceil 3/2 \rceil, \lceil 3/2 \lceil 3/2 \rceil \rceil, \ldots, n'$, where n' is the first value of ℓ that exceeds n, using various types of set I described below. To compute the bridging set, the algorithm maintains the witness matrix with extra polylog factor in the complexity. In [10], there are three ways for selecting the set I. Let $|I| = n^r$ for some r sucn that $0 \le r \le 1$.

(1) Select $9n \ln n/\ell$ vertices from V at random. In this case, it can be shown that the algorithm solves the APSP problem with high probability, i. e., with $1 - 1/n^c$ for some constant $c > 0$, which can be shown to be 3. In other words, I is a strong $\ell/3$-bridging set with high probability. The time T is dominated by (n, m)-Romani. It holds that $T = \tilde{O}(\ell M n^{\omega(1,r,1)})$, since the magnitude of matrix elements can be up to ℓM. Since $m = O(n \ln n/\ell) = n^r$, it holds that $\ell = \tilde{O}(n^{1-r})$, and thus $T = O(Mn^{1-r} n^{\omega(1,r,1)})$. When $M = 1$, this bound on r is $\mu = 0.575$, and thus $T = O(n^{2.575})$. When $M = n^t \ge 1$, the time becomes $O(n^{2+\mu(t)})$, where $t \le 3 - \omega = 0.624$ and $\mu = \mu(t)$ satisfies $\omega(1, \mu, 1) = 1 + 2\mu - t$. It is determined from the best known $\omega(1, \mu, 1)$ and the value of t. As the result is correct with high probability, this is a randomized algorithm.

(2) Consider the case of unit edge costs here. In (1), the computation of witnesses is an extra thing, i. e., not necessary if only shortest distances are needed. To achieve the same complexity in the sense of an exact algorithm, not a randomized one, the computation of witnesses is essential. As mentioned earlier, maintenance of witnesses, that is, matrix W, can be done with an extra polylog factor, meaning the analysis can be focused on Romani within the $\tilde{O}$-notation. Specifically I is selected as an $\ell/3$-bridging set, which is strong with unit edge costs. To compute I as an $O(\ell)$-bridging set, obtain the vertices on the shortest path from i to j for each i and j using the witness matrix W in $O(\ell)$ time. After obtaining those n^2 sets spending $O(\ell n^2)$ time, it is shown in [10] how to obtain a $O(\ell)$-bridging set of $O(n \ln n/\ell)$ size within the same time complexity. The process of obtaining the bridging set must stop at $\ell = n^{1/2}$ as the process is too expensive beyond this point, and thus the same bridging set is used beyond this point. The time before this point is the same as that in (1), and that after this point is $\tilde{O}(n^{2.5})$. Thus, this is a two-phase algorithm.

(3) When edge costs are positive and bounded by $M = n^t > 0$, a similar procedure can be used to compute an $O(\ell)$-bridging set of $O(n \ln n/\ell)$ size in $\tilde{O}(\ell n^2)$ time. Using the bridging set, the APSP problem can be solved in $\tilde{O}(n^{2+\mu(t)})$ time in a similar way to (1). The result can be generalized into the case where edge costs are between $-M$ and M within the same time complexity by modifying the procedure for computing an ℓ-bridging set, provided there is no negative cycle. The details are shown in [10].

Applications

The eccentricity of a vertex v of a graph is the greatest distance from v to any other vertices. The diameter of a graph is the greatest eccentricity of any vertices. In other words, the diameter is the greatest distance between any pair of vertices. If the corresponding APSP problem is solved, the maximum element of the resulting matrix is the diameter.

Open Problems

Two major challenges are stated here among others. The first is to improve the complexity of $\tilde{O}(n^{2.575})$ for the APSP with unit edge costs. The other is to improve the bound of $M < O(n^{0.624})$ for the complexity of the APSP with integer costs up to M to be sub-cubic.

Cross References

► All Pairs Shortest Paths in Sparse Graphs
► Fully Dynamic All Pairs Shortest Paths

Recommended Reading

1. Alon, N., Galil, Z., Margalit, O.: On the exponent of the all pairs shortest path problem. In: Proc. 32th IEEE FOCS, pp. 569–575. IEEE Computer Society, Los Alamitos, USA (1991). Also JCSS **54**, 255–262 (1997)
2. Alon, N., Galil, Z., Margalit, O., Naor, M.: Witnesses for Boolean matrix multiplication and for shortest paths. In: Proc. 33th IEEE FOCS, pp. 417–426. IEEE Computer Society, Los Alamitos, USA (1992)
3. Coppersmith, D., Winograd, S.: Matrix multiplication via arithmetic progressions. J. Symb. Comput. **9**, 251–280 (1990)
4. Coppersmith, D.: Rectangular matrix multiplication revisited. J. Complex. **13**, 42–49 (1997)
5. Huang, X., Pan, V.Y.: Fast rectangular matrix multiplications and applications. J. Complex. **14**, 257–299 (1998)
6. Romani, F.: Shortest-path problem is not harder than matrix multiplications. Info. Proc. Lett. **11**, 134–136 (1980)
7. Seidel, R.: On the all-pairs-shortest-path problem. In: Proc. 24th ACM STOC pp. 745–749. Association for Computing Machinery, New York, USA (1992) Also JCSS **51**, 400–403 (1995)
8. Schönhage, A., Strassen, V.: Schnelle Multiplikation Großer Zahlen. Computing **7**, 281–292 (1971)
9. Takaoka, T.: Sub-cubic time algorithms for the all pairs shortest path problem. Algorithmica **20**, 309–318 (1998)
10. Zwick, U.: All pairs shortest paths using bridging sets and rectangular matrix multiplication. J. ACM **49**(3), 289–317 (2002)

Alternative Performance Measures in Online Algorithms

2000; Koutsoupias, Papadimitriou

ESTEBAN FEUERSTEIN
Department of Computing, University of Buenos Aires, Buenos Aires, Argentina

Keywords and Synonyms

Diffuse adversary model for online algorithms; Comparative analysis for online algorithms

Problem Definition

Even if online algorithms had been studied for around thirty years, the explicit introduction of *competitive analysis* in the seminal papers by Sleator and Tarjan [8] and Manasse, McGeoch and Sleator [6] sparked an extraordinary boom in research about these class of problems and algorithms, so both concepts (online algorithms and competitive analysis) have been strongly related since. However, rather early in its development, some criticism arose regarding the realism and practicality of the model mainly because of its pessimism. That characteristic, in some cases, attempts on the ability of the model to distinguish, between good and bad algorithms. In a 1994 paper called *Beyond competitive analysis* [3], Koutsoupias and Papadimitriou proposed and explored two alternative performance measures for on-line algorithms, both very much related to competitive analysis and yet avoiding the weaknesses that caused the aforementioned criticism. The final version of that work appeared in 2000 [4].

In competitive analysis, the performance of an online algorithm is compared against an all-powerful adversary on a worst-case input. The competitive ratio of an algorithm A is defined as the worst possible ratio

$$R_A = \max_x \frac{A(x)}{opt(x)} ,$$

where x ranges over all possible inputs of the problem and $A(x)$ and $opt(x)$ are respectively the costs of the solutions obtained by algorithm A and the optimum offline algorithm for input x[1]. This notion can be extended to define the competitive ratio of a problem, as the minimum competitive ratio of an algorithm for it, namely

$$R = \min_A R_A = \min_A \max_x \frac{A(x)}{opt(x)} .$$

The main criticism to this approach has been that, with the characteristic pessimism common to all kinds of worst-case analysis, it fails to discriminate between algorithms that could have different performances under different conditions. Moreover, algorithms that "try" to perform well relative to this worst case measure many times fail to behave well in front of many "typical" inputs. This arguments can be more easily contested in the (rare) scenarios where the very strong assumption that *nothing* is known about the distribution of the input holds. But, this is rarely the case in practice.

[1]In this article all problems are assumed to be online minimization problems, therefore the objective is to minimize costs. All the results presented here are valid for online maximization problems with the proper adjustments to the definitions.

The paper by Koutsoupias and Papadimitriou proposes and studies two refinements of competitive analysis which try to overcome all these concerns. The first of them is the *diffuse adversary model*, which points at the cases where something is known about the input: its probabilistic distribution. With this in mind, the performance of an algorithm is evaluated comparing its expected cost with the one of an optimal algorithm for inputs following that distribution.

The second refinement is called *comparative analysis*. This refinement is based on the notion of *information regimes*. According to this, competitive analysis is interpreted as the comparison between two different information regimes, the online and the offline ones. But this vision entails that those information regimes are just particular, extreme cases of a large set of possibilities, among which, for example, the set of algorithms that know in advance some prefix of the awaiting input (finite lookahead algorithms).

Key Results

Diffuse Adversaries

The competitive ratio of an algorithm A against a class Δ of input distributions is the infimum c such that the algorithm is c-competitive when the input is restricted to that class. That happens whenever there exists a constant d such that, for all distributions $D \in \Delta$,

$$\mathcal{E}_D(A(x)) \leq c\mathcal{E}_D(opt(x)) + d\,,$$

where $\mathcal{E}_D$ stands for the mathematical expectation over inputs following distribution D. The competitive ratio $R(\Delta)$ of the class of distributions Δ is the minimum competitive ratio achievable by an online algorithm against Δ.

The model is applied to the traditional Paging problem, for the class of distributions Δ_ϵ. Δ_ϵ is the class that contains all probability distributions such that, given a request sequence and a page p, the probability that the next requested page is p is not more than ϵ. It is shown that the well-known online algorithm LRU achieves the optimal competitive ratio $R(\Delta_\epsilon)$ for all ϵ, that is, it is optimal against any adversary that uses a distribution in this class.

The proof of this result makes strong use of the *work function* concept introduced in [5], that is used as a tool to track the behavior of the optimal offline algorithm and estimate the optimal cost for a sequence of requests, and that of *conservative adversaries*, which are adversaries that assign higher probabilities to pages that have been requested more recently. This kind of adversary is consistent with *locality of reference*, a concept that has been always connected to Paging algorithms and competitive analysis (though in [1] another family of distributions is proposed, and analyzed within this framework, which better captures this notion).

The first result states that, for any adversary $D \in \Delta_\epsilon$, there is a conservative adversary $\hat{D} \in \Delta_\epsilon$ such that the competitive ratio of LRU against $\hat{D}$ is at least the competitive ratio of LRU against D. Then it is shown that for any conservative adversary $D \in \Delta_\epsilon$ against LRU, there is a conservative adversary $D' \in \Delta_\epsilon$, against an on-line algorithm A such that the competitive ratio of LRU against D is at most the competitive ratio of A against D'. In other words, for any ϵ, LRU has the optimal competitive ratio $R(\Delta_\epsilon)$ for the diffuse adversary model. This is the main result in the first part of [4].

The last remaining point refers to the value of the optimal competitive ratio of LRU for the Paging problem. As it is shown, that value is not easy to compute. For the extreme values of ϵ (the cases in which the adversary has complete and almost no control of the input, respectively), $R(\Delta_1) = k$, where k is the size of the cache, and also $\lim_{\epsilon\to 0} R(\Delta_\epsilon) = 1$. Later work by Young [9] allowed to estimate $R(\Delta_\epsilon)$ within (almost) a factor of two. For values of ε around the threshold $1/k$ the optimal ratio is $\Theta(\ln k)$, for values below that threshold the values tend rapidly to $O(1)$, and above it to $\Theta(k)$.

Comparative Analysis

Comparative analysis is a generalization of competitive analysis that allows to compare classes of algorithms, and not just individual algorithms. This new idea may be used to contrast the behaviors of algorithms obeying to arbitrary information regimes. In a few words, an information regime is a class of algorithms that acquire knowledge of the input in the same way, or at similar "rates", so both classes of online and offline algorithms are particular instances of this concept (the former know the input step by step, the latter receive all the information before having to produce any output).

The idea of comparative analysis is to measure the relative quality of two classes of algorithms by the maximum possible quotient of the results obtained by algorithms in each of the classes for the same input.

Formally, if $\mathcal{A}$ and $\mathcal{B}$ are classes of algorithms, the *comparative ratio* $R(\mathcal{A},\mathcal{B})$ is defined as

$$R(\mathcal{A},\mathcal{B}) = \max_{B\in\mathcal{B}} \min_{A\in\mathcal{A}} \max_{x} \frac{A(x)}{B(x)}\,.$$

With this definition, if $\mathcal{B}$ is the class of all algorithms, and $\mathcal{A}$ is the class of on-line algorithms, then the comparative ratio coincides with the competitive ratio.

The concept is illustrated determining how beneficial it can be to allow some *lookahead* to algorithms for *Metrical Task Systems (MTS)*. MTS are an abstract model that has been introduced in [2], and generalizes a wide family of on-line problems, among which Paging, the k-server problem, list accessing, and many other more. In a Metrical Task System a server can travel through the points of a Metric Space (states) while serving a sequence of requests or Tasks. The cost of serving a task depends on the state in which the server is, and the total cost for the sequence is given by the sum of the distance traveled plus the cost of servicing all the tasks. The meaning of the lookahead in this context is that the server can decide where to serve the next task based not only on the past movements and input but also on some fixed number of future requests.

The main result here (apart from the definition of the model itself) is that, for Metrical Task Systems, the Comparative Ratio for the class of online algorithms versus that of algorithms with lookahead l (respectively $\mathcal{L}_0$ and $\mathcal{L}_l$) is not more than $2l + 1$. That is, for this family of problems the benefit obtainable from lookahead is never more than two times the size of the lookahead plus one. The result is completed showing particular cases in which the equality holds.

Finally, for particular Metrical Task System the power of lookahead is shown to be strictly less than that: the last important result of this section shows that for the Paging Problem, the comparative ratio is exactly $min\{l + 1, k\}$, that is, the benefit of using lookahead l is the minimum between the size of the cache and the size of the lookahead window plus one.

Applications

As it is mentioned in the introduction of [4], the ideas presented therein are useful to have a better and more precise analysis of the performance of online algorithms. Also, the diffuse adversary model may prove useful to depict characteristics of the input that are probabilistic in nature (e. g. locality). An example in this direction is a paper by Becchetti [1], that uses a diffuse adversary with the intention of better modeling the locality of reference phenomenon that characterizes practical applications of Paging. In the distributions considered there the probability of requesting a page is also a function of the page's age, and it is shown that the competitive ratio of LRU becomes constant as locality increases.

A different approach is taken however in [7]. There the Paging problem with variable cache size is studied and it is shown that the approach of the expected competitive ratio in the diffuse adversary model can be misleading, while they propose the use of the *average performance ratio* instead.

Open Problems

It is an open problem to determine the exact competitive ratio against a diffuse adversary of known algorithms, for example FIFO, for the Paging problem. FIFO is known to be worse in practice than LRU, so proving that the former is suboptimal for some values of ε would give more support to the model.

An open direction presented in the paper is to consider what they call the *Markov diffuse adversary*, which as it is suggested by the name, refers to an adversary that generates the sequence of requests following a Markov process with output.

The last direction of research suggested is to use the idea of comparative analysis to compare the efficiency of agents or robots with different capabilities (for example with different vision ranges) to perform some tasks (for example construct a plan of the environment).

Cross References

- ▶ List Scheduling
- ▶ Load Balancing
- ▶ Metrical Task Systems
- ▶ Online Interval Coloring
- ▶ Online List Update
- ▶ Packet Switching in Multi-Queue Switches
- ▶ Packet Switching in Single Buffer
- ▶ Paging
- ▶ Robotics
- ▶ Routing
- ▶ Work-Function Algorithm for k Servers

Recommended Reading

1. Becchetti, L.: Modeling locality: A probabilistic analysis of LRU and FWF. In: Proceeding 12th European Symposium on Algorithms (ESA) (2004)
2. Borodin, A., Linial, N., Saks, M.E.: An optimal on-line algorithm for metrical task systems. J. ACM **39**, 745–763 (1992)
3. Koutsoupias, E., Papadimitriou, C.H.: Beyond competitive analysis. In: Proceeding 35th Annual Symposium on Foundations of Computer Science, pp. 394–400, Santa Fe, NM (1994)
4. Koutsoupias, E., Papadimitriou, C.H.: Beyond competitive analysis. SIAM J. Comput. **30**(1), 300–317 (2000)
5. Koutsoupias, E., Papadimitriou, C.H.: On the k-server conjecture. J. ACM **42**(5), 971–983 (1995)
6. Manasse, M.S., McGeoch, L.A., Sleator, D.D.: Competitive algorithms for on-line problems. In: Proceeding 20th Annual ACM Symposium on the Theory of Computing, pp. 322–333, Chicago, IL (1988)

7. Panagiotou, K., Souza, A.: On adequate performance measures for paging. In: Proceeding 38th annual ACM symposium on Theory of computing, STOC 2006
8. Sleator, D.D., Tarjan, R.E.: Amortized efficiency of list update and paging rules. Comm. ACM. **28**, 202–208 (1985)
9. Young, N.E.: On-Line Paging against Adversarially Biased Random Inputs. J. Algorithms **37**, 218 (2000)

Analyzing Cache Misses

2003; Mehlhorn, Sanders

NAILA RAHMAN
Department of Computer Science, University of Leicester, Leicester, UK

Keywords and Synonyms

Cache analysis

Problem Definition

The problem considered here is *multiple sequence access via cache memory*. Consider the following pattern of memory accesses. k sequences of data, which are stored in disjoint arrays and have a total length of N, are accessed as follows:

```
for $t := 1$ to $N$ do
    select a sequence $s_i \in \{1, \dots k\}$
    work on the current element of sequence $s_i$
    advance sequence $s_i$ to the next element.
```

The aim is to obtain exact (not just asymptotic) closed form upper and lower bounds for this problem. Concurrent accesses to multiple sequences of data are ubiquitous in algorithms. Some examples of algorithms which use this paradigm are distribution sorting, k-way merging, priority queues, permuting and FFT. This entry summarises the analyses of this problem in [3,6].

Caches, Models and Cache Analysis

Modern computers have hierarchical memory which consists of registers, one or more levels of caches, main memory and external memory devices such as disks and tapes. Memory size increases but the speed decreases with distance from the CPU. Hierarchical memory is designed to improve the performance of algorithms by exploiting temporal and spatial locality in data accesses.

Caches are modeled as follows. A cache has m *blocks* each of which holds B data elements. The capacity of the cache is $M = mB$. Data is transferred between one level of cache and the next larger and slower memory in blocks of B elements. A cache is organized as $s = m/a$ *sets* where each set consists of a blocks. Memory at address xB, referred to as memory block x can only be placed in a block in set $x \bmod s$. If $a = 1$ the cache is said to be *direct mapped* and if $a = s$ it is said to be *fully associative*.

If memory block x is accessed and it is not in cache then a *cache miss* occurs and the data in memory block x is brought into cache, incurring a performance penalty. In order to accommodate block x, it is assumed that the least recently used (LRU) or the first used (FIFO) block from the cache set $x \bmod s$ is evicted and this is referred to as the *replacement strategy*. Note that a block may be evicted from a set even though there may be unoccupied blocks in other sets.

Cache analysis is performed for the number of cache misses for a problem with N data elements. To read or write N data elements an algorithm must incur $\Omega(N/B)$ cache misses. These are the *compulsory* or *first reference misses*. In the multiple sequence access via cache memory problem, for given values of M and B, one aim is to find the largest k such that there are $O(N/B)$ cache misses for the N data accesses. It is interesting to analyze cache misses for the important case of direct mapped cache and for the general case of set-associative caches.

A large number of algorithms have been designed on the External Memory Model [9] and these algorithms optimize the number of data transfers between main memory and disk. It seems natural to exploit these algorithms to minimize cache misses, but due to the limited associativity of caches this is not straightforward. In the external memory model data transfers are under programmer control and the multiple sequence access problem has a trivial solution. The algorithm simply chooses $k \le M_e/B_e$, where B_e is the block size and M_e is the capacity of the main memory in the external memory model. For $k \le M_e/B_e$ there are $O(N/B_e)$ accesses to external memory. Since caches are hardware controlled the problem becomes nontrivial. For example, consider the case where the starting addresses of $k > a$ equal length sequences map to the ith element of the same set and the sequences are accessed in a round-robin fashion. On a cache with an LRU or FIFO replacement strategy all sequence accesses will result in a cache miss. Such pathological cases can be overcome by randomizing the starting addresses of the sequences.

Related Problems

A very closely related problem is where accesses to the sequences are interleaved with accesses to a small working array. This occurs in applications such as distribution sorting or matrix multiplication.

Caches can emulate external memory with an optimal replacement policy [1,8] however this requires some constant factor more memory. Since the emulation techniques are software controlled and require modification to the algorithm, rather than selection of parameters, they work well for fairly simple algorithms [4].

Key Results

Theorem 1 [3] *Given an a-way set associative cache with m cache blocks, $s = m/a$ cache sets, cache blocks size B, and LRU or FIFO replacement strategy. Let U_a denote the expected number of cache misses in any schedule of N sequential accesses to k sequences with starting addresses that are at least $(a+1)$-wise independent.*

$$U_1 \le k + \frac{N}{B}\left(1 + (B-1)\frac{k}{m}\right) , \tag{1}$$

$$U_1 \ge \frac{N}{B}\left(1 + (B-1)\frac{k-1}{m+k-1}\right) , \tag{2}$$

$$U_a \le k + \frac{N}{B}\left(1 + (B-1)\left(\frac{k\alpha}{m}\right)^a + \frac{1}{m/(k\alpha)-1} + \frac{k-1}{s-1}\right) \quad \text{for } k \le \frac{m}{\alpha} , \tag{3}$$

$$U_a \le k + \frac{N}{B}\left(1 + (B-1)\left(\frac{k\beta}{m}\right)^a + \frac{1}{m/(k\beta)-1}\right) \quad \text{for } k \le \frac{m}{2\beta} , \tag{4}$$

$$U_a \ge \frac{N}{B}\left(1 + (B-1)P_{\text{tail}}\left(k-1, \frac{1}{s}, a\right)\right) - kM , \tag{5}$$

$$U_a \ge \frac{N}{B}\left(1 + (B-1)\left(\frac{(k-a)\alpha}{m}\right)^a \left(1 - \frac{1}{s}\right)^k\right) - kM , \tag{6}$$

where $\alpha = \alpha(a) = a/(a!)^{1/a}$, $P_{\text{tail}}(n,p,a) = \sum_{i \ge a} \binom{n}{i} p^i (1-p)^{n-i}$ is the cumulative binomial probability and $\beta := 1 + \alpha(\lceil ax \rceil)$ where $x = x(a) = \inf\{0 < z < 1 : z + z/\alpha(\lceil az \rceil) = 1\}$.

Here $1 \le \alpha < e$ and $\beta(1) = 2$, $\beta(\infty) = 1 + e \approx 3.71$. This analysis assumes that an adversary schedules the accesses to the sequences. For the lower bound the adversary initially advances sequence s_i for $i = 1 \ldots k$ by X_i elements, where the X_i are chosen uniformly and independently from $\{0, M-1\}$. The adversary then accesses the sequences in a round-robin manner.

The k in the upper bound accounts for a possible extra block that may be accessed due to randomization of the starting addresses. The $-kM$ term in the lower bound accounts for the fact that cache misses can not be counted when the adversary initially winds forwards the sequences.

The bounds are of the form $pN + c$, where c does not depend on N and p is called the *cache miss probability*. Letting $r = k/m$, the ratio between the number of sequences and the number of cache blocks, the bounds for the cache miss probabilities in Theorem 1 become [3]:

$$p_1 \le (1/B)(1 + (B-1)r) , \tag{7}$$

$$p_1 \ge (1/B)\left(1 + (B-1)\frac{r}{1+r}\right) , \tag{8}$$

$$p_a \le (1/B)(1 + (B-1)(r\alpha)^a + r\alpha + ar) \text{ for } r \le \frac{1}{\alpha} , \tag{9}$$

$$p_a \le (1/B)(1 + (B-1)(r\beta)^a + r\beta) \text{ for } r \le \frac{1}{2\beta} , \tag{10}$$

$$p_a \ge (1/B)\left(1 + (B-1)(r\alpha)^a \left(1 - \frac{1}{s}\right)^k\right) . \tag{11}$$

The $1/B$ term accounts for the compulsory or first reference miss, which must be incurred in order to read a block of data from a sequence. The remaining terms account for *conflict misses*, which occur when a block of data is evicted from cache before all its elements have been been scanned. Conflict misses can be reduced by restricting the number of sequences. As r approaches zero the cache miss probabilities approach $1/B$. In general, inequality (4) states that the number of cache misses is $O(N/B)$ if $r \le 1/(2\beta)$ and $(B-1)(r\beta)^a = O(1)$. Both these conditions are satisfied if $k \le m/\max(B^{1/a}, 2\beta)$. So, there are $O(N/B)$ cache misses provided $k = O(m/B^{1/a})$.

The analysis shows that for a direct-mapped cache, where $a = 1$, the upper bound is a factor of $r + 1$ above the lower bound. For $a \ge 2$, the upper bounds and lower bounds are close if $(1 - 1/s)^k \approx$ and $(\alpha + a)r \ll 1$ and both these conditions are satisfied if $k \ll s$.

Rahman and Raman [6] obtain closer upper and lower bounds for average case cache misses assuming the sequences are accessed uniformly randomly on a direct-mapped cache. Sen and Chatterjee [8] also obtain upper and lower bounds assuming the sequences are randomly accessed. Ladner, Fix and LaMarca have analyzed the problem on direct-mapped caches on the independent reference model [2].

Multiple Sequence Access with Additional Working Set

As stated earlier in many applications accesses to sequences are interleaved with accesses to an additional data structure, a *working set*, which determines how a sequence element is to be treated. Assuming that the working set has size at most sB and is stored in contiguous memory locations, the following is an upper bound on the number of cache misses:

Theorem 2 *[3] Let U_a denote the bound on the number of cache misses in Theorem 1 and define $U_0 = N$. With the working set occupying w conflict free memory blocks, the expected number of cache misses arising in the N accesses to the sequence data and any number of accesses to the working set, is bounded by $w + (1 - w/s)U_a + 2(w/s)U_{a-1}$.*

On a direct mapped cache, for $i = 1, \ldots, k$, if sequence i is accessed with probability p_i independently of all previous accesses and is followed by an access to element i of the working set then the following are upper and lower bounds for the number of cache misses:

Theorem 3 *[6] In a direct-mapped cache with m cache blocks each of B elements, if sequence i, for $i = 1, \ldots, k$, is accessed with probability p_i and block j of the working set, for $j = 1, \ldots, k/B$, is accessed with probability P_j then the expected number of cache misses in N sequence accesses is at most $N(p_s + p_w) + k(1 + 1/B)$, where:*

$$p_s \le \frac{1}{B} + \frac{k}{mB} + \frac{B-1}{mB} \sum_{i=1}^{k} \left(\sum_{j=1}^{k/B} \frac{p_i P_j}{p_i + P_j} + \frac{B-1}{B} \sum_{j=1}^{k} \frac{p_i p_j}{p_i + p_j} \right),$$

$$p_w \le \frac{k}{B^2 m} + \frac{B-1}{mB} \sum_{i=1}^{k/B} \sum_{j=1}^{k} \frac{P_i p_j}{P_i + p_j}.$$

Theorem 4 *[6] In a direct-mapped cache with m cache blocks each of B elements, if sequence i, for $i = 1, \ldots, k$, is accessed with probability $p_i \ge 1/m$ then the expected number of cache misses in N sequence accesses is at least $Np_s + k$, where:*

$$p_s \ge \frac{1}{B} + \frac{k(2m-k)}{2m^2} + \frac{k(k-3m)}{2Bm^2} - \frac{1}{2Bm} - \frac{k}{2B^2 m} + \frac{B(k-m)+2m-3k}{Bm^2} \sum_{i=1}^{k} \sum_{j=1}^{k} \frac{(p_i)^2}{p_i + p_j} + \frac{(B-1)^2}{B^3 m^2} \sum_{i=1}^{k} p_i \left[\sum_{j=1}^{k} \frac{p_i(1 - p_i - p_j)}{(p_i + p_j)^2} - \frac{B-1}{2} \sum_{j=1}^{k} \sum_{l=1}^{k} \frac{p_i}{p_i + p_j + p_l - p_j p_l} \right] - O\left(e^{-B}\right).$$

The lower bound ignores the interaction with the working set, since this can only increase the number of cache misses.

In Theorem 3 and Theorem 4 p_s is the probability of a cache miss for a sequence access and in Theorem 3 p_w is the probability of a cache miss for an accesses to the working set.

If the sequences are accessed uniformly randomly, then using Theorem 3 and Theorem 4, the ratio between the upper and lower bound is $3/(3 - r)$, where $r = k/m$. So for uniformly random data the lower bound is within a factor of about 3/2 of the upper bound when $k \le m$ and is much closer when $k \ll m$.

Applications

Numerous algorithms have been developed on the external memory model which access multiple sequences of data, such as merge-sort, distribution sort, priority queues, radix sorting. These analyzes are important as they allow initial parameter choices to be made for cache memory algorithms.

Open Problems

The analyzes assume that the starting addresses of the sequences are randomized and current approaches to allocating random starting addresses waste a lot of virtual address space [3]. An open problem is to find a good online scheme to randomize the starting addresses of arbitrary length sequences.

Experimental Results

The cache model is a powerful abstraction of real caches, however modern computer architectures have complex internal memory hierarchies, with registers, multiple levels of caches and *translation-lookaside-buffers* (TLB). Cache miss penalties are not of the same magnitude as the cost of disk accesses, so an algorithm may perform better by allowing conflict misses to increase in order to reduce computation costs and compulsory misses, by reducing the number of passes over the data. This means that in practice cache analyzes is used to choose an initial value of k which is then fine tuned for the platform and algorithm [4,5,7,10].

For distribution sorting, in [4] a heuristic was considered for selecting k and equations for approximate cache misses were obtained. These equations were shown to be very accurate in practice.

Cross References

- ► Cache-Oblivious Model
- ► Cache-Oblivious Sorting
- ► External Sorting and Permuting
- ► I/O-model

Recommended Reading

1. Frigo, M., Leiserson, C.E., Prokop, H., Ramachandran, S.: Cache-oblivious algorithms. In: Proc. of 40th Annual Symposium on Foundations of Computer Science (FOCS'99), pp. 285–298 IEEE Computer Society, Washington D.C. (1999)
2. Ladner, R.E., Fix, J.D., LaMarca, A.: Cache performance analysis of traversals and random accesses. In: Proc. of 10th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 1999), pp. 613–622 Society for Industrial and Applied Mathematics, Philadelphia (1999)
3. Mehlhorn, K., Sanders, P.: Scanning multiple sequences via cache memory. Algorithmica **35**, 75–93 (2003)
4. Rahman, N., Raman, R.: Adapting radix sort to the memory hierarchy. ACM J. Exp. Algorithmics **6**, Article 7 (2001)
5. Rahman, N., Raman, R.: Analysing cache effects in distribution sorting. ACM J. Exp. Algorithmics **5**, Article 14 (2000)
6. Rahman, N., Raman, R.: Cache analysis of non-uniform distribution sorting algorithms. (2007) http://www.citebase.org/abstract?id=oai:arXiv.org:0706.2839 Accessed 13 August 2007 Preliminary version in: Proc. of 8th Annual European Symposium on Algorithms (ESA 2000). LNCS, vol. 1879, pp. 380–391. Springer, Berlin Heidelberg (2000)
7. Sanders, P.: Fast priority queues for cached memory. ACM J. Exp. Algorithmics **5**, Article 7 (2000)
8. Sen, S., Chatterjee, S.: Towards a theory of cache-efficient algorithms. In: Proc. of 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2000), pp. 829–838. Society for Industrial and Applied Mathematics (2000)
9. Vitter, J.S.: External memory algorithms and data structures: dealing with massive data. ACM Comput. Surv. **33**, 209–271 (2001)
10. Wickremesinghe, R., Arge, L., Chase, J.S., Vitter, J.S.: Efficient sorting using registers and caches. ACM J. Exp. Algorithmics **7**, 9 (2002)

Applications of Geometric Spanner Networks

2002; Gudmundsson, Levcopoulos, Narasimhan, Smid

JOACHIM GUDMUNDSSON[1], GIRI NARASIMHAN[2], MICHIEL SMID[3]

[1] DMiST, National ICT Australia Ltd, Alexandria, Australia

[2] School of Computing and Information Science, Florida International University, Miami, FL, USA

[3] School of Computer Science, Carleton University, Ottawa, ON, Canada

Keywords and Synonyms

Stretch factor

Problem Definition

Given a geometric graph in d-dimensional space, it is useful to preprocess it so that distance queries, exact or approximate, can be answered efficiently. Algorithms that can report distance queries in constant time are also referred to as "distance oracles". With unlimited preprocessing time and space, it is clear that exact distance oracles can be easily designed. This entry sheds light on the design of approximate distance oracles with limited preprocessing time and space for the family of geometric graphs with constant dilation.

Notation and Definitions

If p and q are points in $\mathbb{R}^d$, then the notation $|pq|$ is used to denote the Euclidean distance between p and q; the notation $\delta_G(p, q)$ is used to denote the Euclidean length of a shortest path between p and q in a geometric network G. Given a constant $t > 1$, a graph G with vertex set S is a t-spanner for S if $\delta_G(p, q) \leq t|pq|$ for any two points p and q of S. A t-spanner network is said to have *dilation* (or *stretch factor*) t. A $(1 + \varepsilon)$-approximate shortest path between p and q is defined to be any path in G between p and q having length Δ, where $\delta_G(p, q) \leq \Delta \leq (1 + \varepsilon)\delta_G(p, q)$. For a comprehensive overview of geometric spanners, see the book by Narasimhan and Smid [13].

All networks considered in this entry are simple and undirected. The model of computation used is the traditional algebraic computation tree model with the added power of indirect addressing. In particular, the algorithms presented here do not use the non-algebraic floor function as a unit-time operation. The problem is formalized below.

Problem 1 (Distance Oracle) *Given an arbitrary real constant $\varepsilon > 0$, and a geometric graph G in d-dimensional Euclidean space with constant dilation t, design a data structure that answers $(1 + \varepsilon)$-approximate shortest path length queries in constant time.*

The data structure can also be applied to solve several other problems. These include (a) the problem of reporting approximate distance queries between vertices in a planar polygonal domain with "rounded" obstacles, (b) query versions of *closest pair* problems, and (c) the efficient computation of the approximate dilations of geometric graphs.

Survey of Related Research

The design of efficient data structures for answering distance queries for general (non-geometric) networks was considered by Thorup and Zwick [15] (unweighted general graphs), Baswanna and Sen [3] (weighted general graphs, i. e., arbitrary metrics), and Arikati et al. [2] and Thorup [14] (weighted planar graphs).

For the geometric case, variants of the problem have been considered in a number of papers (for a recent paper see, for example, Chen et al. [5]). Work on the approximate version of these variants can also be found in many articles (for a recent paper see, for example, Agarwal et al. [1]). The focus of this entry are the results reported in the work of Gudmundsson et al. [9,10,11,12].

Key Results

The main result of this entry is the existence of approximate distance oracle data structures for geometric networks with constant dilation (see "Theorem 4" below). As preprocessing, the network is "pruned" so that it only has a linear number of edges. The data structure consists of a series of "cluster graphs" of increasing coarseness each of which helps answer approximate queries for pairs of points with interpoint distances of different scales. In order to pinpoint the appropriate cluster graph to search in for a given query, the data structure uses the bucketing tool described below. The idea of using cluster graphs to speed up geometric algorithms was first introduced by Das and Narasimhan [6] and later used by Gudmundsson et al. [8] to design an efficient algorithm to compute $(1 + \varepsilon)$-spanners. Similar ideas were explored by Gao et al. [7] for applications to the design of mobile networks.

Pruning

If the input geometric network has a superlinear number of edges, then the preprocessing step for the distance oracle data structure involves efficiently "pruning" the network so that it has only a linear number of edges. The pruning may result in a small increase of the dilation of the spanner. The following theorem was proved by Gudmundsson et al. [12].

Theorem 1 *Let $t > 1$ and $\varepsilon' > 0$ be real constants. Let S be a set of n points in $\mathbb{R}^d$, and let $G = (S, E)$ be a t-spanner for S with m edges. There exists an algorithm to compute in $O(m + n \log n)$ time, a $(1 + \varepsilon')$-spanner of G having $O(n)$ edges and whose weight is $O(wt(MST(S)))$.*

The pruning step requires the following technical theorem proved by Gudmundsson et al. [12].

Theorem 2 *Let S be a set of n points in $\mathbb{R}^d$, and let $c \geq 7$ be an integer constant. In $O(n \log n)$ time, it is possible to compute a data structure $D(S)$ consisting of:*

1. *a sequence $L_1, L_2, \ldots, L_\ell$ of real numbers, where $\ell = O(n)$, and*
2. *a sequence $S_1, S_2, \ldots, S_\ell$ of subsets of S satisfying $\sum_{i=1}^{\ell} |S_i| = O(n)$,*

such that the following holds. For any two distinct points p and q of S, it is possible to compute in $O(1)$ time an index i with $1 \leq i \leq \ell$ and two points x and y in S_i such that (a) $L_i/n^{c+1} \leq |xy| < L_i$, and (b) both $|px|$ and $|qy|$ are less than $|xy|/n^{c-2}$.

Despite its technical nature, the above theorem is of fundamental importance to this work. In particular, it helps to deal with networks where the interpoint distances are not confined to a polynomial range, i. e., there are pairs of points that are very close to each other and very far from each other.

Bucketing

Since the model of computation assumed here does not allow the use of floor functions, an important component of the algorithm is a "bucketing tool" that allows (after appropriate preprocessing) constant-time computation of a quantity referred to as BINDEX, which is defined to be the floor of the logarithm of the interpoint distance between any pair of input points.

Theorem 3 *Let S be a set of n points in $\mathbb{R}^d$ that are contained in the hypercube $(0, n^k)^d$, for some positive integer constant k, and let ε be a positive real constant. The set S can be preprocessed in $O(n \log n)$ time into a data structure of size $O(n)$, such that for any two points p and q of S, with $|pq| \geq 1$, it is possible to compute in constant time the quantity $BIndex_\varepsilon(p, q) = \lfloor \log_{1+\varepsilon} |pq| \rfloor$.*

The constant-time computation mentioned in Theorem 3 is achieved by reducing the problem to one of answering least common ancestor queries for pairs of nodes in a tree, a problem for which constant-time solutions were devised most recently by Bender and Farach-Colton [4].

Main Results

Using the bucketing and the pruning tools, and using the algorithms described by Gudmundsson et al. [11], the following theorem can be proved.

Theorem 4 *Let $t > 1$ and $\varepsilon > 0$ be real constants. Let S be a set of n points in $\mathbb{R}^d$, and let $G = (S, E)$ be a t-spanner for S with m edges. The graph G can be preprocessed into a data structure of size $O(n \log n)$ in time $O(m + n \log n)$, such that for any pair of query points $p, q \in S$, it is possible to compute a $(1+\varepsilon)$-approximation of the shortest-path distance in G between p and q in $O(1)$ time. Note that all the big-Oh notations hide constants that depend on d, t and ε.*

Additionally, if the traditional algebraic model of computation (without indirect addressing) is assumed, the following weaker result can be proved.

Theorem 5 *Let S be a set of n points in $\mathbb{R}^d$, and let $G = (S, E)$ be a t-spanner for S, for some real constant $t > 1$, having m edges. Assuming the algebraic model of computation, in $O(m \log\log n + n \log^2 n)$ time, it is possible to preprocess G into a data structure of size $O(n \log n)$, such that for any two points p and q in S, a $(1 + \varepsilon)$-approximation of the shortest-path distance in G between p and q can be computed in $O(\log\log n)$ time.*

Applications

As mentioned earlier, the data structure described above can be applied to several other problems. The first application deals with reporting distance queries for a planar domain with polygonal obstacles. The domain is further constrained to be t-rounded, which means that the length of the shortest obstacle-avoiding path between any two points in the input point set is at most t times the Euclidean distance between them. In other words, the visibility graph is required to be a t-spanner for the input point set.

Theorem 6 *Let $\mathcal{F}$ be a t-rounded collection of polygonal obstacles in the plane of total complexity n, where t is a positive constant. One can preprocess $\mathcal{F}$ in $O(n \log n)$ time into a data structure of size $O(n \log n)$ that can answer obstacle-avoiding $(1+\varepsilon)$-approximate shortest path length queries in time $O(\log n)$. If the query points are vertices of $\mathcal{F}$, then the queries can be answered in O(1) time.*

The next application of the distance oracle data structure includes query versions of *closest pair* problems, where the queries are confined to specified subset(s) of the input set.

Theorem 7 *Let $G = (S, E)$ be a geometric graph on n points and m edges, such that G is a t-spanner for S, for some constant $t > 1$. One can preprocess G in time $O(m+n \log n)$ into a data structure of size $O(n \log n)$ such that given a query subset S' of S, a $(1 + \varepsilon)$-approximate closest pair in S' (where distances are measured in G) can be computed in time $O(|S'| \log |S'|)$.*

Theorem 8 *Let $G = (S, E)$ be a geometric graph on n points and m edges, such that G is a t-spanner for S, for some constant $t > 1$. One can preprocess G in time $O(m+n \log n)$ into a data structure of size $O(n \log n)$ such that given two disjoint query subsets X and Y of S, a $(1 + \varepsilon)$-approximate bichromatic closest pair (where distances are measured in G) can be computed in time $O((|X| + |Y|) \log(|X| + |Y|))$.*

The last application of the distance oracle data structure includes the efficient computation of the approximate dilations of geometric graphs.

Theorem 9 *Given a geometric graph on n vertices with m edges, and given a constant C that is an upper bound on the dilation t of G, it is possible to compute a $(1 + \varepsilon)$-approximation to t in time $O(m + n \log n)$.*

Open Problems

Two open problems remain unanswered.

1. Improve the space utilization of the distance oracle data structure from $O(n \log n)$ to $O(n)$.
2. Extend the approximate distance oracle data structure to report not only the approximate distance, but also the approximate shortest path between the given query points.

Cross References

▶ All Pairs Shortest Paths in Sparse Graphs
▶ All Pairs Shortest Paths via Matrix Multiplication
▶ Geometric Spanners
▶ Planar Geometric Spanners
▶ Sparse Graph Spanners
▶ Synchronizers, Spanners

Recommended Reading

1. Agarwal, P.K., Har-Peled, S., Karia, M.: Computing approximate shortest paths on convex polytopes. In: Proceedings of the 16th ACM Symposium on Computational Geometry, pp. 270–279. ACM Press, New York (2000)
2. Arikati, S., Chen, D.Z., Chew, L.P., Das, G., Smid, M., Zaroliagis, C.D.: Planar spanners and approximate shortest path queries among obstacles in the plane. In: Proceedings of the 4th Annual European Symposium on Algorithms. Lecture Notes in

Computer Science, vol. 1136, Berlin, pp. 514–528. Springer, London (1996)
3. Baswana, S., Sen, S.: Approximate distance oracles for unweighted graphs in $\tilde{O}(n^2)$ time. In: Proceedings of the 15th ACM-SIAM Symposium on Discrete Algorithms, pp. 271–280. ACM Press, New York (2004)
4. Bender, M.A., Farach-Colton, M.: The LCA problem revisited. In: Proceedings of the 4th Latin American Symposium on Theoretical Informatics. Lecture Notes in Computer Science, vol. 1776, Berlin, pp. 88–94. Springer, London (2000)
5. Chen, D.Z., Daescu, O., Klenk, K.S.: On geometric path query problems. Int. J. Comput. Geom. Appl. **11**, 617–645 (2001)
6. Das, G., Narasimhan, G.: A fast algorithm for constructing sparse Euclidean spanners. Int. J. Comput. Geom. Appl. **7**, 297–315 (1997)
7. Gao, J., Guibas, L.J., Hershberger, J., Zhang, L., Zhu, A.: Discrete mobile centers. Discrete Comput. Geom. **30**, 45–63 (2003)
8. Gudmundsson, J., Levcopoulos, C., Narasimhan, G.: Fast greedy algorithms for constructing sparse geometric spanners. SIAM J. Comput. **31**, 1479–1500 (2002)
9. Gudmundsson, J., Levcopoulos, C., Narasimhan, G., Smid, M.: Approximate distance oracles for geometric graphs. In: Proceedings of the 13th ACM-SIAM Symposium on Discrete Algorithms, pp. 828–837. ACM Press, New York (2002)
10. Gudmundsson, J., Levcopoulos, C., Narasimhan, G., Smid, M.: Approximate distance oracles revisited. In: Proceedings of the 13th International Symposium on Algorithms and Computation. Lecture Notes in Computer Science, vol. 2518, Berlin, pp. 357–368. Springer, London (2002)
11. Gudmundsson, J., Levcopoulos, C., Narasimhan, G., Smid, M.: Approximate distance oracles for geometric spanners, ACM Trans. Algorithms (2008). To Appear
12. Gudmundsson, J., Narasimhan, G., Smid, M.: Fast pruning of geometric spanners. In: Proceedings of the 22nd Symposium on Theoretical Aspects of Computer Science. Lecture Notes in Computer Science, vol. 3404, Berlin, pp. 508–520. Springer, London (2005)
13. Narasimhan, G., Smid, M.: Geometric Spanner Networks, Cambridge University Press, Cambridge, UK (2007)
14. Thorup, M.: Compact oracles for reachability and approximate distances in planar digraphs. J. ACM **51**, 993–1024 (2004)
15. Thorup, M., Zwick, U.: Approximate distance oracles. In: Proceedings of the 33rd Annual ACM Symposium on the Theory of Computing, pp. 183–192. ACM Press, New York (2001)

Approximate Dictionaries

2002; Buhrman, Miltersen, Radhakrishnan, Venkatesh

VENKATESH SRINIVASAN
Department of Computer Science, University of Victoria, Victoria, BC, Canada

Keywords and Synonyms

Static membership; Approximate membership

Problem Definition

The Problem and the Model

A static data structure problem consists of a set of data D, a set of queries Q, a set of answers A, and a function $f : D \times Q \to A$. The goal is to store the data succinctly so that any query can be answered with only a few probes to the data structure. *Static membership* is a well-studied problem in data structure design [1,4,7,8,12,13,16].

Definition 1 (Static Membership) In the static membership problem, one is given a subset S of at most n keys from a universe $U = \{1, 2, \dots, m\}$. The task is to store S so that queries of the form "Is u in S?" can be answered by making few accesses to the memory.

A natural and general model for studying any data structure problem is the *cell probe model* proposed by Yao [16].

Definition 2 (Cell Probe Model) An (s, w, t) cell probe scheme for a static data structure problem $f : D \times Q \to A$ has two components: a storage scheme and a query scheme. The storage scheme stores the data $d \in D$ as a table $T[d]$ of s cells, each cell of word size w bits. The storage scheme is deterministic. Given a query $q \in Q$, the query scheme computes $f(d, q)$ by making at most t probes to $T[d]$, where each probe reads one cell at a time, and the probes can be adaptive. In a deterministic cell probe scheme, the query scheme is deterministic. In a randomized cell probe scheme, the query scheme is randomized and is allowed to err with a small probability.

Buhrman et al. [2] study the complexity of the static membership problem in the *bitprobe* model. The bitprobe model is a variant of the cell probe model in which each cell holds just a single bit. In other words, the word size w is 1. Thus, in this model, the query algorithm is given bitwise access to the data structure. The study of the membership problem in the bitprobe model was initiated by Minsky and Papert in their book *Perceptrons* [12]. However, they were interested in *average*-case upper bounds for this problem, while this work studies *worst*-case bounds for the membership problem.

Observe that if a scheme is required to store sets of size at most n, then it must use at least $\lceil \log \sum_{i \le n} \binom{m}{i} \rceil$ number of bits. If $n \le m^{1-\Omega(1)}$, this implies that the scheme must store $\Omega(n \log m)$ bits (and therefore use $\Omega(n)$ cells). The goal in [2] is to obtain a scheme that answers queries uses only a constant number of bitprobes and at the same time uses a table of $O(n \log m)$ bits.

Related Work

The static membership problem has been well studied in the cell probe model, where each cell is capable of holding one element of the universe. That is, $w = O(\log m)$. In a seminal paper, Fredman et al. [8] proposed a scheme for the static membership problem in the cell probe model with word size $O(\log m)$ that used a constant number of probes and a table of size $O(n)$. This scheme will be referred to as the FKS scheme. Thus, up to constant factors, the FKS scheme uses optimal space and number of cell probes. In fact, Fiat et al. [7], Brodnik and Munro [1], and Pagh [13] obtain schemes that use space (in bits) that is within a small additive term of $\lceil \log \sum_{i \le n} \binom{m}{i} \rceil$ and yet answer queries by reading at most a constant number of cells. Despite all these fundamental results for the membership problem in the cell probe model, very little was known about the bitprobe complexity of static membership prior to the work in [2].

Key Results

Buhrman et al. investigate the complexity of the static membership problem in the bitprobe model. They study

- Two-sided error randomized schemes that are allowed to err on positive instances as well as negative instances (that is, these schemes can say 'No' with a small probability when the query element u is in the set S and 'Yes' when it is not);
- One-sided error randomized schemes where the errors are restricted to negative instances alone (that is, these schemes never say 'No' when the query element u is in the set S);
- Deterministic schemes in which no errors are allowed.

The main techniques used in [2] are based on two-colorings of special set systems that are related to the r-cover-free family of sets considered in [3,5,9]. The reader is referred to [2] for further details.

Randomized Schemes with Two-Sided Error

The main result in [2] shows that there are randomized schemes that use *just one bitprobe* and yet use space close to the information theoretic lower bound of $\Omega(n \log m)$ bits.

Theorem 1 *For any $0 < \epsilon \le \frac{1}{4}$, there is a scheme for storing subsets S of size at most n of a universe of size m using $O(\frac{n}{\epsilon^2} \log m)$ bits so that any membership query "Is $u \in S$?" can be answered with an error probability of at most ϵ by a randomized algorithm that probes the memory at just one location determined by its coin tosses and the query element u.*

Note that randomization is allowed only in the query algorithm. It is still the case that for each set S, there is exactly one associated data structure $T(S)$. It can be shown that deterministic schemes that answer queries using a single bitprobe need m bits of storage (see the remarks following Theorem 4). Theorem 1 shows that, by allowing randomization, this bound (for constant ϵ) can be reduced to $O(n \log m)$ bits. This space is within a constant factor of the information theoretic bound for n sufficiently small. Yet the randomized scheme answers queries using a single bitprobe.

Unfortunately, the construction above does not permit us to have subconstant error probability and still use optimal space. Is it possible to improve the result of Theorem 1 further and design such a scheme? [2] shows that this is not possible: if ϵ is made subconstant, then the scheme must use more than $n \log m$ space.

Theorem 2 *Suppose $\frac{n}{m^{1/3}} \le \epsilon \le \frac{1}{4}$. Then, any two-sided ϵ-error randomized scheme that answers queries using one bitprobe must use space $\Omega(\frac{n}{\epsilon \log(1/\epsilon)} \log m)$.*

Randomized Schemes with One-Sided Error

Is it possible to have any savings in space if the query scheme is expected to make only one-sided errors? The following result shows it is possible if the error is allowed only on negative instances.

Theorem 3 *For any $0 < \epsilon \le \frac{1}{4}$, there is a scheme for storing subsets S of size at most n of a universe of size m using $O((\frac{n}{\epsilon})^2 \log m)$ bits so that any membership query "Is $u \in S$?" can be answered with error probability at most ϵ by a randomized algorithm that makes a single bitprobe to the data structure. Furthermore, if $u \in S$, the probability of error is 0.*

Though this scheme does not operate with optimal space, it still uses significantly less space than a bitvector. However, the dependence on n is quadratic, unlike in the two-sided scheme where it was linear. [2] shows that this scheme is essentially optimal: there is necessarily a quadratic dependence on $\frac{n}{\epsilon}$ for any scheme with one-sided error.

Theorem 4 *Suppose $\frac{n}{m^{1/3}} \le \epsilon \le \frac{1}{4}$. Consider the static membership problem for sets S of size at most n from a universe of size m. Then, any scheme with one-sided error ϵ that answers queries using at most one bitprobe must use $\Omega(\frac{n^2}{\epsilon^2 \log(n/\epsilon)} \log m)$ bits of storage.*

Remark One could also consider one-probe, one-sided error schemes that only make errors on positive instances. That is, no error is made for query elements *not* in the set S.

In this case, [2] shows that randomness does not help at all: such a scheme must use m bits of storage.

The following result shows that the space requirement can be reduced further in one-sided error schemes if more probes are allowed.

Theorem 5 *Suppose $0 < \delta < 1$. There is a randomized scheme with one-sided error $n^{-\delta}$ that solves the static membership problem using $O(n^{1+\delta} \log m)$ bits of storage and $O(\frac{1}{\delta})$ bitprobes.*

Deterministic Schemes

In contrast to randomized schemes, Buhrman et al. show that deterministic schemes exhibit a time-space tradeoff behavior.

Theorem 6 *Suppose a deterministic scheme stores subsets of size n from a universe of size m using s bits of storage and answers membership queries with t bitprobes to memory. Then, $\binom{m}{n} \leq \max_{i \leq nt} \binom{2s}{i}$.*

This tradeoff result has an interesting consequence. Recall that the FKS hashing scheme is a data structure for storing sets of size at most n from a universe of size m using $O(n \log m)$ bits, so that membership queries can be answered using $O(\log m)$ bitprobes. As a corollary of the tradeoff result, [2] shows that the FKS scheme makes an optimal number of bitprobes, within a constant factor, for this amount of space.

Corollary 1 *Let $\epsilon > 0, c \geq 1$ be any constants. There is a constant $\delta > 0$ so that the following holds. Let $n \leq m^{1-\epsilon}$ and let a scheme for storing sets of size at most n of a universe of size m as data structures of at most $cn \log m$ bits be given. Then, any deterministic algorithm answering membership queries using this structure must make at least $\delta \log m$ bitprobes in the worst case.*

From Theorem 6 it also follows that any deterministic scheme that answers queries using t bitprobes must use space at least $ntm^{\Omega(1/t)}$ in the worst case. The final result shows the existence of schemes which almost match the lower bound.

Theorem 7

1. *There is a nonadaptive scheme that stores sets of size at most n from a universe of size m using $O(ntm^{\frac{2}{t+1}})$ bits and answers queries using $2t + 1$ bitprobes. This scheme is nonexplicit.*
2. *There is an explicit adaptive scheme that stores sets of size at most n from a universe of size m using $O(m^{1/t} n \log m)$ bits and answers queries using $O(\log n + \log\log m) + t$ bitprobes.*

Applications

The results in [2] have interesting connections to questions in coding theory and communication complexity. In the framework of coding theory, the results in [2] can be viewed as constructing locally decodable source codes, analogous to the locally decodable channel codes of [10]. Theorems 1–4 can also be viewed as giving tight bounds for the following communication complexity problem (as pointed out in [11]): Alice gets $u \in \{1, \ldots, m\}$, Bob gets $S \subseteq \{1, \ldots, m\}$ of size at most n, and Alice sends a single message to Bob after which Bob announces whether $u \in S$. See [2] for further details.

Recommended Reading

1. Brodnik, A., Munro, J.I.: Membership in constant time and minimum space. In: Lecture Notes in Computer Science, vol. 855, pp. 72–81, Springer, Berlin (1994). Final version: Membership in Constant Time and Almost-Minimum Space. SIAM J. Comput. **28**(5), 1627–1640 (1999)
2. Buhrman, H., Miltersen, P.B., Radhakrishnan, J., Venkatesh, S.: Are bitvectors optimal? SIAM J. Comput. **31**(6), 1723–1744 (2002)
3. Dyachkov, A.G., Rykov, V.V.: Bounds on the length of disjunctive codes. Problemy Peredachi Informatsii **18**(3), 7–13 (1982)
4. Elias, P., Flower, R.A.: The complexity of some simple retrieval problems. J. Assoc. Comput. Mach. **22**, 367–379 (1975)
5. Erdös, P., Frankl, P., Füredi, Z.: Families of finite sets in which no set is covered by the union of r others. Isr. J. Math. **51**, 79–89 (1985)
6. Fiat, A., Naor, M.: Implicit $O(1)$ probe search. SIAM J. Comput. **22**, 1–10 (1993)
7. Fiat, A., Naor, M., Schmidt, J.P., Siegel, A.: Non-oblivious hashing. J. Assoc. Comput. Mach. **31**, 764–782 (1992)
8. Fredman, M.L., Komlós, J., Szemerédi, E.: Storing a sparse table with $O(1)$ worst case access time. J. Assoc. Comput. Mach. **31**(3), 538–544 (1984)
9. Füredi, Z.: On r-cover-free families. J. Comb. Theory, Series A **73**, 172–173 (1996)
10. Katz, J., Trevisan, L.: On the efficiency of local decoding procedures for error-correcting codes. In: Proceedings of STOC'00, pp. 80–86
11. Miltersen, P.B., Nisan, N., Safra, S., Wigderson, A.: On data structures and asymmetric communication complexity. J. Comput. Syst. Sci. **57**, 37–49 (1998)
12. Minsky, M., Papert, S.: Perceptrons. MIT Press, Cambridge (1969)
13. Pagh, R.: Low redundancy in static dictionaries with $O(1)$ lookup time. In: Proceedings of ICALP '99. LNCS, vol. 1644, pp. 595–604. Springer, Berlin (1999)
14. Ruszinkó, M. On the upper bound of the size of r-cover-free families. J. Comb. Theory, Ser. A **66**, 302–310 (1984)
15. Ta-Shma, A.: Explicit one-probe storing schemes using universal extractors. Inf. Proc. Lett. **83**(5), 267–274 (2002)
16. Yao, A.C.C.: Should tables be sorted? J. Assoc. Comput. Mach. **28**(3), 615–628 (1981)

Approximate Dictionary Matching

► Dictionary Matching and Indexing (Exact and with Errors)

Approximate Maximum Flow Construction

► Randomized Parallel Approximations to Max Flow

Approximate Membership

► Approximate Dictionaries

Approximate Nash Equilibrium

► Non-approximability of Bimatrix Nash Equilibria

Approximate Periodicities

► Approximate Tandem Repeats

Approximate Regular Expression Matching

1995; Wu, Manber, Myers

GONZALO NAVARRO
Department of Computer Science, University of Chile, Santiago, Chile

Keywords and Synonyms

Regular expression matching allowing errors or differences

Problem Definition

Given a *text string* $T = t_1 t_2 \ldots t_n$ and a *regular expression* R of length m denoting language $\mathcal{L}(R)$, over an alphabet Σ of size σ, and given a *distance function* among strings d and a *threshold* k, the *approximate regular expression matching (AREM)* problem is to find all the text positions that finish a so-called *approximate occurrence* of R in T, that is, compute the set $\{j, \exists i, 1 \le i \le j, \exists P \in \mathcal{L}(R), d(P, t_i \ldots t_j) \le k\}$. T, R, and k are given together, whereas the algorithm can be tailored for a specific d.

This entry focuses on the so-called *weighted edit distance*, which is the minimum sum of weights of a sequence of operations converting one string into the other. The operations are insertions, deletions, and substitutions of characters. The weights are positive real values associated to each operation and characters involved. The weight of deleting a character c is written $w(c \to \epsilon)$, that of inserting c is written $w(\epsilon \to c)$, and that of substituting c by $c' \neq c$ is written $w(c \to c')$. It is assumed $w(c \to c) = 0$ for all $c \in \Sigma \cup \epsilon$ and the triangle inequality, that is, $w(x \to y) + w(y \to z) \ge w(x \to z)$ for any $x, y, z \in \Sigma \cup \{\epsilon\}$. As the distance may be asymmetric, it is also fixed that that $d(A, B)$ is the cost of converting A into B. For simplicity and practicality $m = o(n)$ is assumed in this entry.

Key Results

The most versatile solution to the problem [3] is based on a graph model of the distance computation process. Assume the regular expression R is converted into a nondeterministic finite automaton (NFA) with $O(m)$ states and transitions using Thompson's method [8]. Take this automaton as a directed graph $G(V, E)$ where edges are labeled by elements in $\Sigma \cup \{\epsilon\}$. A directed and weighted graph $\mathcal{G}$ is built to solve the AREM problem. $\mathcal{G}$ is formed by putting $n + 1$ copies of G, $G_0, G_1, \ldots, G_n$, and connecting them with weights so that the distance computation reduces to finding shortest paths in $\mathcal{G}$.

More formally, the nodes of $\mathcal{G}$ are $\{v_i, v \in V, 0 \le i \le n\}$, so that v_i is the copy of node $v \in V$ in graph G_i. For each edge $u \xrightarrow{c} v$ in E, $c \in \Sigma \cup \{\epsilon\}$, the following edges are added to graph $\mathcal{G}$:

$$\begin{array}{lll} u_i \to v_i\,, & \text{with weight } w(c \to \epsilon)\,, & 0 \le i \le n\,, \\ u_i \to u_{i+1}\,, & \text{with weight } w(\epsilon \to t_{i+1})\,, & 0 \le i < n\,, \\ u_i \to v_{i+1}\,, & \text{with weight } w(c \to t_{i+1})\,, & 0 \le i < n\,. \end{array}$$

Assume for simplicity that G has initial state s and a unique final state f (this can always be arranged). As defined, the shortest path in $\mathcal{G}$ from s_0 to f_n gives the smallest distance between T and a string in $\mathcal{L}(R)$. In order to adapt the graph to the AREM problem, the weights of the edges between s_i and s_{i+1} are modified to be zero.

Then, the AREM problem is reduced to computing shortest paths. It is not hard to see that $\mathcal{G}$ can be topologically sorted so that all the paths to nodes in G_i are computed before all those to G_{i+1}. This way, it is not hard to solve this shortest path problem in $O(mn \log m)$ time and $O(m)$ space. Actually, if one restricts the problem to the particular case of *network expressions*, which are regular expressions without Kleene closure, then G has no loops and the shortest path computation can be done in $O(mn)$ time, and even better on average [2].

The most delicate part in achieving $O(mn)$ time for general regular expressions [3] is to prove that, given the types of loops that arise in the NFAs of regular expressions, it is possible to compute the distances correctly within each G_i by (a) computing them in a topological order of G_i without considering the *back edges* introduced by Kleene closures; (b) updating path costs by using the back edges once; (c) updating path costs once more in topological order ignoring back edges again.

Theorem 1 (Myers and Miller 1989 [3]) *There exists an $O(mn)$ worst-case time solution to the AREM problem under weighted edit distance.*

It is possible to do better when the weights are integer-valued, by exploiting the unit-cost RAM model through a four-Russian technique [10]. The idea is as follows. Take a small subexpression of R, which produces an NFA that will translate into a small subgraph of each G_i. At the time of propagating path costs within this automaton, there will be a counter associated to each node (telling the current shortest path from s_0). This counter can be reduced to a number in $[0, k+1]$, where $k+1$ means "more than k". If the small NFA has r states, $r\lceil\log_2(k+2)\rceil$ bits are needed to fully describe the counters of the corresponding subgraph of G_i. Moreover, given an initial set of values for the counters, it is possible to precompute all the propagation that will occur within the same subgraph of G_i, in a table having $2^{r\lceil\log_2(k+2)\rceil}$ entries, one per possible configuration of counters. It is sufficient that $r < \alpha \log_{k+2} n$ for some $\alpha < 1$ to make the construction and storage cost of those tables $o(n)$. With the help of those tables, all the propagation within the subgraph can be carried out in constant time. Similarly, the propagation of costs to the same subgraph at G_{i+1} can also be precomputed in tables, as it depends only on the current counters in G_i and on text character t_{i+1}, for which there are only σ alternatives.

Now, take all the subtrees of R of maximum size not exceeding r and preprocess them with the technique above. Convert each such subtree into a leaf in R labeled by a special character a_A, associated to the corresponding small NFA A. Unless there are consecutive Kleene closures in R, which can be simplified as $R^{**} = R^*$, the size of R after this transformation is $O(m/r)$. Call R' the transformed regular expression. One essentially applies the technique of Theorem 1 to R', taking care of how to deal with the special leaves that correspond to small NFAs. Those leaves are converted by Thompson's construction into two nodes linked by an edge labeled a_A. When the path cost propagation process reaches the source node of an edge labeled a_A with cost c, one must update the counter of the initial state of NFA A to c (or $k+1$ if $c > k$). One then uses the four-Russians table to do all the cost propagation within A in constant time, and finally obtain, at the counter of the final state of A, the new value for the target node of the edge labeled a_A in the top-level NFA. Therefore, all the edges (normal and special) of the top-level NFA can be traversed in constant time, so the costs at G_i can be obtained in $O(mn/r)$ time using Theorem 1. Now one propagates the costs to G_{i+1}, using the four-Russians tables to obtain the current counter values of each subgraph A in G_{i+1}.

Theorem 2 (Wu et al. 1995 [10]) *There exists an $O(n + mn/\log_{k+2} n)$ worst-case time solution to the AREM problem under weighted edit distance if the weights are integer numbers.*

Applications

The problem has applications in computational biology, to find certain types of motifs in DNA and protein sequences. See [1] for a more detailed discussion. In particular, PROSITE patterns are limited regular expressions rather popular to search protein sequences. PROSITE patterns can be searched for with faster algorithms in practice [7]. The same occurs with other classes of complex patterns [6] and network expressions [2].

Open Problems

The worst-case complexity of the AREM problem is not fully understood. It is of course $\Omega(n)$, which has been achieved for $m\log(k+2) = O(\log n)$, but it is not known how much can this be improved.

Experimental Results

Some recent experiments are reported in [5]. For small m and k, and assuming all the weights are 1 (except $w(c \to c) = 0$), bit-parallel algorithms of worst-case complexity $O(kn(m/\log n)^2)$ [4,9] are the fastest (the second is able to skip some text characters, depending on R). For arbitrary integer weights, the best choice is a more complex bit-parallel algorithm [5]; or the four-Russians based one [10] for larger m and k. The original algorithm [3] is slower but it is the only one supporting arbitrary weights.

URL to Code

Well-known packages offering efficient AREM (for simplified weight choices) are *agrep* [9] (http://webglimpse.net/download.html, top-level subdirectory `agrep/`) and *nrgrep* [4] (http://www.dcc.uchile.cl/~gnavarro/software). For biological applications, *anrep* [2] (http://www.cs.arizona.edu/people/gene/CODE/anrep.tar.Z) matches sequences of approximate network expressions with arbitrary weights and a specified gap length between each network expression and the next.

Cross References

► Regular Expression Matching is the simplified case where exact matching with strings in $\mathcal{L}(R)$ is sought.
► Sequential Approximate String Matching is a simplification of this problem, and the relation between graph G here and matrix C there should be apparent.

Recommended Reading

1. Gusfield, D.: Algorithms on strings, trees and sequences. Cambridge University Press, Cambridge (1997)
2. Myers, E.W.: Approximate matching of network expressions with spacers. J. Comput. Biol. **3**(1), 33–51 (1996)
3. Myers, E.W., Miller, W.: Approximate matching of regular expressions. Bullet. Math. Biol. **51**, 7–37 (1989)
4. Navarro, G.: Nr-grep: a fast and flexible pattern matching tool. Softw. Pr. Exp. **31**, 1265–1312 (2001)
5. Navarro, G.: Approximate regular expression searching with arbitrary integer weights. Nord. J. Comput. **11**(4), 356–373 (2004)
6. Navarro, G., Raffinot, M.: Flexible Pattern Matching in Strings – Practical on-line search algorithms for texts and biological sequences. Cambridge University Press, Cambridge (2002)
7. Navarro, G., Raffinot, M.: Fast and simple character classes and bounded gaps pattern matching, with applications to protein searching. J. Comput. Biol. **10**(6), 903–923 (2003)
8. Thompson, K.: Regular expression search algorithm. Commun. ACM **11**(6), 419–422 (1968)
9. Wu, S., Manber, U.: Fast text searching allowing errors. Commun. ACM **35**(10), 83–91 (1992)
10. Wu, S., Manber, U., Myers, E.W.: A subquadratic algorithm for approximate regular expression matching. J. Algorithms **19**(3), 346–360 (1995)

Approximate Repetitions

► Approximate Tandem Repeats

Approximate Tandem Repeats

2001; Landau, Schmidt, Sokol
2003; Kolpakov, Kucherov

GREGORY KUCHEROV[1], DINA SOKOL[2]
[1] LIFL and INRIA, Villeneuve d'Ascq, France
[2] Department of Computer and Information Science, Brooklyn College of CUNY, Brooklyn, NY, USA

Keywords and Synonyms

Approximate repetitions; Approximate periodicities

Problem Definition

Identification of periodic structures in words (variants of which are known as *tandem repeats, repetitions, powers* or *runs*) is a fundamental algorithmic task (see entry ► Squares and Repetitions). In many practical applications, such as DNA sequence analysis, considered repetitions admit a certain variation between copies of the repeated pattern. In other words, repetitions under interest are *approximate tandem repeats* and not necessarily exact repeats only.

The simplest instance of an approximate tandem repeat is an *approximate square*. An approximate square in a word w is a subword uv, where u and v are within a given distance k according to some distance measure between words, such as Hamming distance or edit (also called Levenstein) distance. There are several ways to define approximate tandem repeats as successions of approximate squares, i. e. to generalize to the approximate case the notion of arbitrary periodicity (see entry ► Squares and Repetitions). In this entry, we discuss three different definitions of approximate tandem repeats. The first two are built upon the Hamming distance measure, and the third one is built upon the edit distance.

Let $h(\cdot,\cdot)$ denote the Hamming distance between two words of equal length.

Definition 1 A word $r[1..n]$ is called a K-*repetition* of period p, $p \le n/2$, iff $h(r[1..n-p], r[p+1..n]) \le K$.

Equivalently, a word $r[1..n]$ is a K-repetition of period p, if the number of mismatches, i. e. the number of i such that $r[i] \ne r[i+p]$, is at most K. For example, *ataa atta ctta ct* is a 2-repetition of period 4. *atc atc atc atg atg atg atg atg* is a 1-repetition of period 3 but *atc atc atc att atc atc atc att* is not.

Definition 2 A word $r[1..n]$ is called a K-*run*, of period p, $p \le n/2$, iff for every $i \in [1..n-2p+1]$, we have $h(r[i..i+p-1], r[i+p, i+2p-1]) \le K$.

A K-run can be seen as a sequence of approximate squares uv such that $|u| = |v| = p$ and u and v differ by at most K mismatches. The total number of mismatches in a K-run is not bounded.

Let $ed(\cdot,\cdot)$ denote the edit distance between two strings.

Definition 3 A word r is a K-*edit* repeat if it can be partitioned into consecutive subwords, $r = v'w_1w_2\dots w_\ell v''$,

$\ell \geq 2$, such that

$$ed(v', w'_1) + \sum_{i=1}^{\ell-1} ed(w_i, w_{i+1}) + ed(w''_\ell, v'') \leq K\,,$$

where w'_1 is some suffix of w_1 and w''_ℓ is some prefix of w_ℓ.

A K-edit repeat is a sequence of "evolving" copies of a pattern such that there are at most K insertions, deletions, and mismatches, overall, between all consecutive copies of the repeat. For example, the word $r = caagct\,cagct\,ccgct$ is a 2-edit repeat.

When looking for tandem repeats occurring in a word, it is natural to consider *maximal* repeats. Those are the repeats extended to the right and left as much as possible provided that the corresponding definition is still verified. Note that the notion of maximality applies to K-repetitions, to K-runs, and to K-edit repeats.

Under the Hamming distance, K-runs provide the weakest "reasonable" definition of approximate tandem repeats, since it requires that every square it contains cannot contain more than K mismatch errors, which seems to be a minimal reasonable requirement. On the other hand, K-repetition is the strongest such notion as it limits by K the *total* number of mismatches. This provides an additional justification that finding these two types of repeats is important as they "embrace" other intermediate types of repeats. Several intermediate definitions have been discussed in [10, Section 5].

In general, each K-repetition is a part of a K-run of the same period and every K-run is the union of all K-repetitions it contains. Observe that a K-run can contain as many as a linear number of K-repetitions with the same period. For example, the word $(000\,100)^n$ of length $6n$ is a 1-run of period 3, which contains $(2n - 1)$ 1-repetitions. In general, a K-run r contains $(s - K + 1)$ K-repetitions of the same period, where s is the number of mismatches in r.

Example 1 The following Fibonacci word contains three 3-runs of period 6. They are shown in regular font, in positions aligned with their occurrences. Two of them are identical, and contain each four 3-repetitions, shown in italic for the first run only. The third run is a 3-repetition in itself.

```
010010  100100  101001  010010  010100  1001

 10010  100100  101001
 10010  100100  10
  0010  100100  101
    10  100100  10100
     0  100100  101001
                  1001  010010  010100  1
                            10  010100  1001
```

Key Results

Given a word w of length n and an integer K, it is possible to find all K-runs, K-repetitions, and K-edit repeats within w in the following time and space bounds:

***K*-runs** can be found in time $O(nK \log K + S)$ (S the output size) and working space $O(n)$ [10],

***K*-repetitions** can be found in time $O(nK \log K + S)$ and working space $O(n)$ [10],

***K*-edit repeats** can be found in time $O(nK \log K \log(n/K) + S)$ and working space $O(n + K^2)$ [14,19].

All three algorithms are based on similar algorithmic tools that generalize corresponding techniques for the exact case [4,15,16] (see [11] for a systematic presentation). The first basic tool is a generalization of *longest extension functions* [16] that, in the case of Hamming distance, can be exemplified as follows. Given a word w, we want to compute, for each position p and each $k \leq K$, the quantity $\max\{j | h(w[1..j], w[p..p + j - 1]) \leq k\}$. Computing all those values can be done in time $O(nK)$ using a method based on the suffix tree and the computation of *lowest common ancestor* described in [7].

The second tool is the Lempel–Ziv factorization used in the well-known compression method. Different variants of the Lempel–Ziv factorization of a word can be computed in linear time [7,18].

The algorithm for computing K-repetitions from [10] can be seen as a direct generalization of the algorithm for computing maximal repetitions (runs) in the exact case [8,15]. Although based on the same basic tools and ideas, the algorithm [10] for computing K-runs is much more involved and uses a complex "bootstrapping" technique for assembling runs from smaller parts.

The algorithm for finding the K-edit repeats uses both the recursive framework and the idea of the *longest extension functions* of [16]. The longest common extensions, in this case, allow up to K edit operations. Efficient methods for computing these extensions are based upon a combination of the results of [12] and [13]. The K-edit repeats are derived by combining the longest common extensions computed in the forward direction with those computed in the reverse direction.

Applications

Tandemly repeated patterns in DNA sequences are involved in various biological functions and are used in different practical applications.

Tandem repeats are known to be involved in regulatory mechanisms, e. g. to act as binding sites for regulatory proteins. Tandem repeats have been shown to be associated with recombination hot-spots in higher organisms. In bacteria, a correlation has been observed between certain tandem repeats and virulence and pathogenicity genes.

Tandem repeats are responsible for a number of inherited diseases, especially those involving the central nervous system. Fragile X syndrome, Kennedy disease, myotonic dystrophy, and Huntington's disease are among the diseases that have been associated with triplet repeats.

Examples of different genetic studies illustrating above-mentioned biological roles of tandem repeats can be found in introductive sections of [1,6,9]. Even more than just genomic elements associated with various biological functions, tandem repeats have been established to be a fundamental mutational mechanism in genome evolution [17].

A major practical application of short tandem repeats is based on the inter-individual variability in copy number of certain repeats occurring at a single loci. This feature makes tandem repeats a convenient tool for genetic profiling of individuals. The latter, in turn, is applied to pedigree analysis and establishing phylogenetic relationships between species, as well as to forensic medicine [3].

Open Problems

The definition of K-edit repeats is similar to that of K-repetitions (for the Hamming distance case). It would be interesting to consider other definitions of maximal repeats over the edit distance. For example, a definition similar to the K-run would allow up to K edits between each pair of neighboring periods in the repeat. Other possible definitions would allow K errors between *any* pair of copies of a repeat, or between *all pairs* of copies, or between some *consensus* and each copy.

In general, a *weighted* edit distance scheme is necessary for biological applications. Known algorithms for tandem repeats based on a weighted edit distance scheme are not feasible, and thus only heuristics are currently used.

URL to Code

The algorithms described in this entry have been implemented for DNA sequences, and are publicly available. The Hamming distance algorithms (K-runs and K-repetitions) are part of the `mreps` software package, available at http://bioinfo.lifl.fr/mreps/ [9]. The K-edit repeats software, *TRED*, is available at http://www.sci.brooklyn.cuny.edu/~sokol/tandem [19]. The implementations of the algorithms are coupled with postprocessing filters, necessary due to the nature of biological sequences.

In practice, software based on heuristic and statistical methods is largely used. Among them, TRF (http://tandem.bu.edu/trf/trf.html) [1] is the most popular program used by the bioinformatics community. Other programs include ATRHunter (http://bioinfo.cs.technion.ac.il/atrhunter/) [20], TandemSWAN (http://strand.imb.ac.ru/swan/) [2]. STAR (http://atgc.lirmm.fr/star/) [5] is another software, based on an information-theoretic approach, for computing approximate tandem repeats of a pre-specified pattern.

Cross References

► Squares and Repetitions

Acknowledgments

This work was supported in part by the National Science Foundation Grant DB&I 0542751.

Recommended Reading

1. Benson, G.: Tandem Repeats Finder: a program to analyze DNA sequences. Nucleic Acids Res. **27**, 573–580 (1999)
2. Boeva, V.A., Régnier, M., Makeev, V.J.: SWAN: searching for highly divergent tandem repeats in DNA sequences with the evaluation of their statistical significance. Proceedings of JOBIM 2004, Montreal, Canada, p. 40 (2004)
3. Butler, J.M.: Forensic DNA Typing: Biology and Technology Behind STR Markers. Academic Press (2001)
4. Crochemore, M.: Recherche linéaire d'un carré dans un mot. Comptes Rendus Acad. Sci. Paris Sér. I Math. **296**, 781–784 (1983)
5. Delgrange, O., Rivals, E.: STAR – an algorithm to Search for Tandem Approximate Repeats. Bioinform. **20**, 2812–2820 (2004)
6. Gelfand, Y., Rodriguez, A., Benson, G.: TRDB – The Tandem Repeats Database. Nucl. Acids Res. **35**(suppl. 1), D80–D87 (2007)
7. Gusfield, D.: Algorithms on Strings, Trees, and Sequences. Cambridge University Press (1997)
8. Kolpakov, R., Kucherov, G.: Finding maximal repetitions in a word in linear time. In: 40th Symp. Foundations of Computer Science (FOCS), pp. 596–604. IEEE Computer Society Press (1999)
9. Kolpakov, R., Bana, G., Kucherov, G.: `mreps`: efficient and flexible detection of tandem repeats in DNA. Nucl. Acids Res. **31**(13), 3672–3678 (2003)
10. Kolpakov, R., Kucherov, G.: Finding approximate repetitions under Hamming distance. Theoret. Comput. Sci. **33**(1), 135–156, (2003)
11. Kolpakov, R., Kucherov, G.: Identification of periodic structures in words. In: Berstel, J., Perrin, D. (eds.) Applied combinatorics on words. Encyclopedia of Mathematics and its Applications. Lothaire books, vol. 104, pp. 430–477. Cambridge University Press (2005)
12. Landau, G.M., Vishkin, U.: Fast string matching with k differences. J. Comput. Syst. Sci. **37**(1), 63–78 (1988)
13. Landau, G.M., Myers, E.W., Schmidt, J.P.: Incremental string comparison. SIAM J. Comput. **27**(2), 557–582 (1998)

14. Landau, G.M., Schmidt, J.P., Sokol, D.: An algorithm for approximate tandem repeats. J. Comput. Biol. **8**, 1–18 (2001)
15. Main, M.: Detecting leftmost maximal periodicities. Discret. Appl. Math. **25**, 145–153 (1989)
16. Main, M., Lorentz, R.: An $O(n \log n)$ algorithm for finding all repetitions in a string. J. Algorithms **5**(3), 422–432 (1984)
17. Messer, P.W., Arndt, P.F.: The majority of recent short DNA insertions in the human genome are tandem duplications. Mol. Biol. Evol. **24**(5), 1190–7 (2007)
18. Rodeh, M., Pratt, V., Even, S.: Linear algorithm for data compression via string matching. J. Assoc. Comput. Mach. **28**(1), 16–24 (1981)
19. Sokol, D., Benson, G., Tojeira, J.: Tandem repeats over the edit distance. Bioinform. **23**(2), e30–e35 (2006)
20. Wexler, Y., Yakhini, Z., Kashi, Y., Geiger, D.: Finding approximate tandem repeats in genomic sequences. J. Comput. Biol. **12**(7), 928–42 (2005)

Approximating Metric Spaces by Tree Metrics

1996; Bartal, Fakcharoenphol, Rao, Talwar
2004; Bartal, Fakcharoenphol, Rao, Talwar

JITTAT FAKCHAROENPHOL[1], SATISH RAO[2], KUNAL TALWAR[3]
[1] Department of Computer Engineering, Kasetsart University, Bangkok, Thailand
[2] Computer Science Division, University of California at Berkeley, Berkeley, CA, USA
[3] Microsoft Research, Silicon Valley Campus, Mountain View, CA, USA

Keywords and Synonyms

Embedding general metrics into tree metrics

Problem Definition

This problem is to construct a random tree metric that probabilistically approximates a given arbitrary metric well. A solution to this problem is useful as the first step for numerous approximation algorithms because usually solving problems on trees is easier than on general graphs. It also finds applications in on-line and distributed computation.

It is known that tree metrics approximate general metrics badly, e. g., given a cycle C_n with n nodes, any tree metric approximating this graph metric has distortion $\Omega(n)$ [17]. However, Karp [15] noticed that a random spanning tree of C_n approximates the distances between any two nodes in C_n well in expectation. Alon, Karp, Peleg, and West [1] then proved a bound of $\exp(O(\sqrt{\log n \log\log n}))$ on an average distortion for approximating any graph metric with its spanning tree.

Bartal [2] formally defined the notion of probabilistic approximation.

Notations

A graph $G = (V, E)$ with an assignment of non-negative weights to the edges of G defines a metric space (V, d_G) where for each pair $u, v \in V$, $d_G(u, v)$ is the shortest path distance between u and v in G. A metric (V, d) is a *tree metric* if there exists some tree $T = (V', E')$ such that $V \subseteq V'$ and for all $u, v \in V$, $d_T(u, v) = d(u, v)$. The metric (V, d) is also called a metric induced by T.

Given a metric (V, d), a distribution $\mathcal{D}$ over tree metrics over V α-*probabilistically approximates* d if every tree metric $d_T \in \mathcal{D}$, $d_T(u, v) \geq d(u, v)$ and $\mathrm{E}_{d_T \in \mathcal{D}}[d_T(u, v)] \leq \alpha \cdot d(u, v)$, for every $u, v \in V$. The quantity α is referred to as the *distortion* of the approximation.

Although the definition of probabilistic approximation uses a distribution $\mathcal{D}$ over tree metrics, one is interested in a procedure that constructs a random tree metric distributed according to $\mathcal{D}$, i. e., an algorithm that produces a random tree metric that probabilistically approximates a given metric. The problem can be formally stated as follows.

Problem (APPROX-TREE)

INPUT: *a metric (V, d)*

OUTPUT: *a tree metric (V, d_T) sampled from a distribution $\mathcal{D}$ over tree metrics that α-probabilistically approximates (V, d).*

Bartal then defined a class of tree metrics, called hierarchically well-separated trees (HST), as follows. A k*-hierarchically well-separated tree* (k-HST) is a rooted weighted tree satisfying two properties: the edge weight from any node to each of its children is the same, and the edge weights along any path from the root to a leaf are decreasing by a factor of at least k. These properties are important to many approximation algorithms.

Bartal showed that any metric on n points can be probabilistically approximated by a set of k-HST's with $O(\log^2 n)$ distortion, an improvement from $\exp(O(\sqrt{\log n \log\log n}))$ in [1]. Later Bartal [3], following the same approach as in Seymour's analysis on the Feedback Arc Set problem [18], improved the distortion down to $O(\log n \log\log n)$. Using a rounding procedure of Calinescu, Karloff, and Rabani [5], Fakcharoenphol, Rao, and Talwar [9] devised an algorithm that, in expectation, produces a tree with $O(\log n)$ distortion. This bound is tight up to a constant factor.

Key Results

A tree metric is closely related to graph decomposition. The randomized rounding procedure of Calinescu, Karloff, and Rabani [5] for the 0-extension problem decomposes a graph into pieces with bounded diameter, cutting each edge with probability proportional to its length and a ratio between the numbers of nodes at certain distances. Fakcharoenphol, Rao, and Talwar [9] used the CKR rounding procedure to decompose the graph recursively and obtained the following theorem.

Theorem 1 *Given an n-point metric (V, d), there exists a randomized algorithm, which runs in time $O(n^2)$, that samples a tree metric from the distribution $\mathcal{D}$ over tree metrics that $O(\log n)$-probabilistically approximates (V, d). The tree is also a 2-HST.*

The bound in Theorem 1 is tight, as Alon et al. [1] proved the bound of an $\Omega(\log n)$ distortion when (V, d) is induced by a grid graph. Also note that it is known (as folklore) that even embedding a line metric onto a 2-HST requires distortion $\Omega(\log n)$.

If the tree is required to be a k-HST, one can apply the result of Bartal, Charikar, and Raz [4] which states that any 2-HST can be $O(k/\log k)$-probabilistically approximated by k-HST, to obtain an expected distortion of $O(k \log n/\log k)$.

Finding a distribution of tree metrics that probabilistically approximates a given metric has a dual problem that is to find a single tree T with small average weighted stretch. More specifically, given weight c_{uv} on edges, find a tree metric d_T such that for all $u, v \in V d_T(u, v) \geq d(u, v)$ and $\sum_{u,v \in V} c_{uv} \cdot d_T(u, v) \leq \alpha \sum_{u,v \in V} c_{uv} \cdot d(u, v)$.

Charikar, Chekuri, Goel, Guha, and Plotkin [6] showed how to find a distribution of $O(n \log n)$ tree metrics that α-probabilistically approximates a given metric, provided that one can solve the dual problem. The algorithm in Theorem 1 can be derandomized by the method of conditional expectation to find the required tree metric with $\alpha = O(\log n)$. Another algorithm based on modified region growing techniques is presented in [9], and independently by Bartal.

Theorem 2 *Given an n-point metric (V, d), there exists a polynomial-time deterministic algorithm that finds a distribution $\mathcal{D}$ over $O(n \log n)$ tree metrics that $O(\log n)$-probabilistically approximates (V, d).*

Note that the tree output by the algorithm contains Steiner nodes, however Gupta [10] showed how to find another tree metric without Steiner nodes while preserving all distances within a constant factor.

Applications

Metric approximation by random trees has applications in on-line and distributed computation, since randomization works well against oblivious adversaries, and trees are easy to work with and maintain. Alon et al. [1] first used tree embedding to give a competitive algorithm for the k-server problem. Bartal [3] noted a few problems in his paper: metrical task system, distributed paging, distributed k-server problem, distributed queuing, and mobile user.

After the paper by Bartal in 1996, numerous applications in approximation algorithms have been found. Many approximation algorithms work for problems on tree metrics or HST metrics. By approximating general metrics with these metrics, one can turn them into algorithms for general metrics, while, usually, losing only a factor of $O(\log n)$ in the approximation factors. Sample problems are metric labeling, buy-at-bulk network design, and group Steiner trees. Recent applications include an approximation algorithm to the Unique Games [12], information network design [13], and oblivious network design [11].

The SIGACT News article [8] is a review of the metric approximation by tree metrics with more detailed discussion on developments and techniques. See also [3,9], for other applications.

Open Problems

Given a metric induced by a graph, some application, e. g., solving a certain class of linear systems, does not only require a tree metric, but a tree metric induced by a spanning tree of the graph. Elkin, Emek, Spielman, and Teng [7] gave an algorithm for finding a spanning tree with average distortion of $O(\log^2 n \log \log n)$. It remains open if this bound is tight.

Cross References

▶ Metrical Task Systems
▶ Sparse Graph Spanners

Recommended Reading

1. Alon, N., Karp, R.M., Peleg, D., West, D.: A graph-theoretic game and its application to the k-server problem. SIAM J. Comput. **24**, 78–100 (1995)
2. Bartal, Y.: Probabilistic approximation of metric spaces and its algorithmic applications. In: FOCS '96: Proceedings of the 37th Annual Symposium on Foundations of Computer Science, Washington, DC, USA, IEEE Computer Society, pp. 184–193 (1996)
3. Bartal, Y.: On approximating arbitrary metrices by tree metrics. In: STOC '98: Proceedings of the thirtieth annual ACM sympo-

sium on Theory of computing, pp. 161–168. ACM Press, New York (1998)
4. Bartal, Y., Charikar, M., Raz, D.: Approximating min-sum k-clustering in metric spaces. In: STOC '01: Proceedings of the thirty-third annual ACM symposium on Theory of computing, pp. 11–20. ACM Press, New York (2001)
5. Calinescu, G., Karloff, H., Rabani, Y.: Approximation algorithms for the 0-extension problem. In: SODA '01: Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms, Philadelphia, PA, USA, Society for Industrial and Applied Mathematics, pp. 8–16. (2001)
6. Charikar, M., Chekuri, C., Goel, A., Guha, S.: Rounding via trees: deterministic approximation algorithms for group steiner trees and k-median. In: STOC '98: Proceedings of the thirtieth annual ACM symposium on Theory of computing, pp. 114–123. ACM Press, New York (1998)
7. Elkin, M., Emek, Y., Spielman, D.A., Teng, S.-H.: Lower-stretch spanning trees. In: STOC '05: Proceedings of the thirty-seventh annual ACM symposium on Theory of computing, pp. 494–503. ACM Press, New York (2005)
8. Fakcharoenphol, J., Rao, S., Talwar, K.: Approximating metrics by tree metrics. SIGACT News **35**, 60–70 (2004)
9. Fakcharoenphol, J., Rao, S., Talwar, K.: A tight bound on approximating arbitrary metrics by tree metrics. J. Comput. Syst. Sci. **69**, 485–497 (2004)
10. Gupta, A.: Steiner points in tree metrics don't (really) help. In: SODA '01: Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms, Philadelphia, PA, USA, Society for Industrial and Applied Mathematics, pp. 220–227. (2001)
11. Gupta, A., Hajiaghayi, M.T., Räcke, H.: Oblivious network design. In: SODA '06: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm, pp. 970–979. ACM Press, New York (2006)
12. Gupta, A., Talwar, K.: Approximating unique games. In: SODA '06: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm, New York, NY, USA, pp. 99–106. ACM Press, New York (2006)
13. Hayrapetyan, A., Swamy, C., Tardos, É.: Network design for information networks. In: SODA '05: Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms, Philadelphia, PA, USA, Society for Industrial and Applied Mathematics, pp. 933–942. (2005)
14. Indyk, P., Matousek, J.: Low-distortion embeddings of finite metric spaces. In: Goodman, J.E., O'Rourke, J. (eds.) Handbook of Discrete and Computational Geometry. CRC Press, Inc., Chap. 8 (2004), To appear
15. Karp, R.: A 2*k*-competitive algorithm for the circle. Manuscript (1989)
16. Matousek, J.: Lectures on Discrete Geometry. Springer, New York (2002)
17. Rabinovich, Y., Raz, R.: Lower bounds on the distortion of embedding finite metric spaces in graphs. Discret. Comput. Geom. **19**, 79–94 (1998)
18. Seymour, P.D.: Packing directed circuits fractionally. Combinatorica **15**, 281–288 (1995)

Approximation Algorithm

► Knapsack

Approximation Algorithm Design

► Steiner Trees

Approximation Algorithms

► Graph Bandwidth

Approximation Algorithms in Planar Graphs

► Approximation Schemes for Planar Graph Problems

Approximations of Bimatrix Nash Equilibria

2003; Lipton, Markakis, Mehta
2006; Daskalaskis, Mehta, Papadimitriou
2006; Kontogiannis, Panagopoulou, Spirakis

Spyros Kontogiannis[1], Panagiota Panagopoulou[2], Paul Spirakis[3]
[1] Computer Science Department, University of Ioannina, Ioannina, Greece
[2] Research Academic Computer Technology Institute, Patras, Greece
[3] Computer Engineering and Informatics Research and Academic Computer Technology Institute, Patras University, Patras, Greece

Keywords and Synonyms

ϵ-Nash equilibria; ϵ-Well-supported Nash equilibria

Problem Definition

Nash [14] introduced the concept of Nash equilibria in non-cooperative games and proved that any game possesses at least one such equilibrium. A well-known algorithm for computing a Nash equilibrium of a 2-player game is the Lemke-Howson algorithm [12], however it has exponential worst-case running time in the number of available pure strategies [16].

Recently, Daskalakis et al. [5] showed that the problem of computing a Nash equilibrium in a game with 4 or more players is PPAD-complete; this result was later extended to games with 3 players [8]. Eventually, Chen and Deng [3] proved that the problem is PPAD-complete for 2-player games as well.

This fact emerged the computation of *approximate* Nash equilibria. There are several versions of approximate Nash equilibria that have been defined in the literature; however the focus of this entry is on the notions of ϵ-Nash equilibrium and ϵ-well-supported Nash equilibrium. An ϵ-Nash equilibrium is a strategy profile such that no deviating player could achieve a payoff higher than the one that the specific profile gives her, plus ϵ. A stronger notion of approximate Nash equilibria is the *ϵ-well-supported Nash equilibria*; these are strategy profiles such that each player plays only approximately best-response pure strategies with non-zero probability.

Notation

For a $n \times 1$ vector $\mathbf{x}$ denote by $x_1, \ldots, x_n$ the components of $\mathbf{x}$ and by $\mathbf{x}^{\mathrm{T}}$ the transpose of $\mathbf{x}$. Denote by $\mathbf{e_i}$ the column vector with a 1 at the ith coordinate and 0 elsewhere. For an $n \times m$ matrix A, denote a_{ij} the element in the i-th row and j-th column of A. Let $\mathbb{P}^n$ be the set of all probability vectors in n dimensions: $\mathbb{P}^n = \left\{\mathbf{z} \in \mathbb{R}^n_{\geq 0} : \sum_{i=1}^{n} z_i = 1\right\}$.

Bimatrix Games

Bimatrix games [18] are a special case of 2-player games such that the payoff functions can be described by two real $n \times m$ matrices A and B. The n rows of A, B represent the *action set* of the first player (the *row player*) and the m columns represent the action set of the second player (the *column player*). Then, when the row player chooses action i and the column player chooses action j, the former gets payoff a_{ij} while the latter gets payoff b_{ij}. Based on this, bimatrix games are denoted by $\Gamma = \langle A, B \rangle$.

A *strategy* for a player is any probability distribution on her set of actions. Therefore, a strategy for the row player can be expressed as a probability vector $\mathbf{x} \in \mathbb{P}^n$ while a strategy for the column player can be expressed as a probability vector $\mathbf{y} \in \mathbb{P}^m$. Each extreme point $\mathbf{e_i} \in \mathbb{P}^n$ ($\mathbf{e_j} \in \mathbb{P}^m$) that corresponds to the strategy assigning probability 1 to the i-th row (j-th column) is called a *pure strategy* for the row (column) player. A *strategy profile* $(\mathbf{x}, \mathbf{y})$ is a combination of (mixed in general) strategies, one for each player. In a given strategy profile $(\mathbf{x}, \mathbf{y})$ the players get *expected payoffs* $\mathbf{x}^{\mathrm{T}} A\mathbf{y}$ (row player) and $\mathbf{x}^{\mathrm{T}} B\mathbf{y}$ (column player).

If both payoff matrices belong to $[0,1]^{m \times n}$ then the game is called a $[0,1]$-bimatrix (or else, *positively normalized*) game. The special case of bimatrix games in which all elements of the matrices belong to $\{0,1\}$ is called a $\{0,1\}$-bimatrix (or else, *win-lose*) game. A bimatrix game $\langle A, B \rangle$ is called *zero sum* if $B = -A$.

Approximate Nash Equilibria

Definition 1 (ϵ-Nash equilibrium) For any $\epsilon > 0$ a strategy profile $(\mathbf{x}, \mathbf{y})$ is an *ϵ-Nash equilibrium* for the $n \times m$ bimatrix game $\Gamma = \langle A, B \rangle$ if

1. For all pure strategies $i \in \{1, \ldots, n\}$ of the row player, $\mathbf{e_i}^{\mathrm{T}} A\mathbf{y} \leq \mathbf{x}^{\mathrm{T}} A\mathbf{y} + \epsilon$ and
2. For all pure strategies $j \in \{1, \ldots, m\}$ of the column player, $\mathbf{x}^{\mathrm{T}} B\mathbf{e_j} \leq \mathbf{x}^{\mathrm{T}} B\mathbf{y} + \epsilon$.

Definition 2 (ϵ-well-supported Nash equilibrium) For any $\epsilon > 0$ a strategy profile $(\mathbf{x}, \mathbf{y})$ is an *ϵ-well-supported Nash equilibrium* for the $n \times m$ bimatrix game $\Gamma = \langle A, B \rangle$ if

1. For all pure strategies $i \in \{1, \ldots, n\}$ of the row player,
$$x_i > 0 \Rightarrow \mathbf{e_i}^{\mathrm{T}} A\mathbf{y} \geq \mathbf{e_k}^{\mathrm{T}} A\mathbf{y} - \epsilon \quad \forall k \in \{1, \ldots, n\}$$
2. For all pure strategies $j \in \{1, \ldots, m\}$ of the column player,
$$y_j > 0 \Rightarrow \mathbf{x}^{\mathrm{T}} B\mathbf{e_j} \geq \mathbf{x}^{\mathrm{T}} B\mathbf{e_k} - \epsilon \quad \forall k \in \{1, \ldots, m\}\,.$$

Note that both notions of approximate equilibria are defined with respect to an additive error term ϵ. Although (exact) Nash equilibria are known not to be affected by any positive scaling, it is important to mention that approximate notions of Nash equilibria are indeed affected. Therefore, the commonly used assumption in the literature when referring to approximate Nash equilibria is that the bimatrix game is positively normalized, and this assumption is adopted in the present entry.

Key Results

The work of Althöfer [1] shows that, for *any* probability vector $\mathbf{p}$ there exists a probability vector $\hat{\mathbf{p}}$ with logarithmic supports, so that for a fixed matrix C, $\max_j \left|\mathbf{p}^{\mathrm{T}} C\mathbf{e_j} - \hat{\mathbf{p}}^{\mathrm{T}} C\mathbf{e_j}\right| \leq \epsilon$, for any constant $\epsilon > 0$. Exploiting this fact, the work of Lipton, Markakis and Mehta [13], shows that, for any bimatrix game and for any *constant* $\epsilon > 0$, there exists an ϵ-Nash equilibrium with only logarithmic support (in the number n of available pure strategies). Consider a bimatrix game $\Gamma = \langle A, B \rangle$ and let $(\mathbf{x}, \mathbf{y})$ be a Nash equilibrium for Γ. Fix a positive integer k and form a multiset S_1 by sampling k times from the set of pure strategies of the row player, independently at random according to the distribution $\mathbf{x}$. Similarly, form a multiset S_2 by sampling k times from set of pure strategies of the column player according to $\mathbf{y}$. Let $\hat{\mathbf{x}}$ be the mixed strategy for the row player that assigns probability $1/k$ to each member of S_1 and 0 to all other pure strategies, and let $\hat{\mathbf{y}}$

be the mixed strategy for the column player that assigns probability $1/k$ to each member of S_2 and 0 to all other pure strategies. Then $\hat{\mathbf{x}}$ and $\hat{\mathbf{y}}$ are called *k-uniform* [13] and the following holds:

Theorem 1 ([13]) *For any Nash equilibrium* $(\mathbf{x}, \mathbf{y})$ *of a* $n \times n$ *bimatrix game and for every* $\epsilon > 0$, *there exists, for every* $k \geq (12 \ln n)/\epsilon^2$, *a pair of k-uniform strategies* $\hat{\mathbf{x}}, \hat{\mathbf{y}}$ *such that* $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ *is an* ϵ*-Nash equilibrium.*

This result directly yields a quasi-polynomial ($n^{O(\ln n)}$) algorithm for computing such an approximate equilibrium. Moreover, as pointed out in [1], no algorithm that examines supports smaller than about $\ln n$ can achieve an approximation better than 1/4.

Theorem 2 ([4]) *The problem of computing a* $1/n^{\Theta(1)}$*-Nash equilibrium of a* $n \times n$ *bimatrix game is* PPAD*-complete.*

Theorem 2 asserts that, unless PPAD $\subseteq$ P, there exists no fully polynomial time approximation scheme for computing equilibria in bimatrix games. However, this does not rule out the existence of a polynomial approximation scheme for computing an ϵ-Nash equilibrium when ϵ is an absolute constant, or even when $\epsilon = \Theta\left(1/poly(\ln n)\right)$. Furthermore, as observed in [4], if the problem of finding an ϵ-Nash equilibrium were PPAD-complete when ϵ is an absolute constant, then, due to Theorem 1, all PPAD problems would be solved in quasi-polynomial time, which is unlikely to be the case.

Two concurrent and independent works [6,10] were the first to make progress in providing ϵ-Nash equilibria and ϵ-well-supported Nash equilibria for bimatrix games and some *constant* $0 < \epsilon < 1$. In particular, the work of Kontogiannis, Panagopoulou and Spirakis [10] proposes a simple linear-time algorithm for computing a 3/4-Nash equilibrium for any bimatrix game:

Theorem 3 ([10]) *Consider any* $n \times m$ *bimatrix game* $\Gamma = \langle A, B\rangle$ *and let* $a_{i_1,j_1} = \max_{i,j} a_{ij}$ *and* $b_{i_2,j_2} = \max_{i,j} b_{ij}$. *Then the pair of strategies* $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ *where* $\hat{x}_{i_1} = \hat{x}_{i_2} = \hat{y}_{j_1} = \hat{y}_{j_2} = 1/2$ *is a 3/4-Nash equilibrium for* Γ.

The above technique can be extended so as to obtain a parametrized, stronger approximation:

Theorem 4 ([10]) *Consider a* $n \times m$ *bimatrix game* $\Gamma = \langle A, B\rangle$. *Let* λ_1^* (λ_2^*) *be the minimum, among all Nash equilibria of* Γ, *expected payoff for the row (column) player and let* $\lambda = \max\{\lambda_1^*, \lambda_2^*\}$. *Then, there exists a* $(2 + \lambda)/4$*-Nash equilibrium that can be computed in time polynomial in n and m.*

The work of Daskalakis, Mehta and Papadimitriou [6] provides a simple algorithm for computing a 1/2-Nash equilibrium: Pick an arbitrary row for the row player, say row i. Let $j = \arg\max_{j'} b_{ij'}$. Let $k = \arg\max_{k'} a_{k'j}$. Thus, j is a best-response column for the column player to the row i, and k is a best-response row for the row player to the column j. Let $\hat{\mathbf{x}} = 1/2\mathbf{e_i} + 1/2\mathbf{e_k}$ and $\hat{\mathbf{y}} = \mathbf{e_j}$, i. e., the row player plays row i or row k with probability 1/2 each, while the column player plays column j with probability 1. Then:

Theorem 5 ([6]) *The strategy profile* $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ *is a* 1/2*-Nash equilibrium.*

A polynomial construction (based on Linear Programming) of a 0.38-Nash equilibrium is presented in [7].

For the more demanding notion of well-supported approximate Nash equilibrium, Daskalakis, Mehta and Papadimitriou [6] propose an algorithm, which, under a quite interesting and plausible graph theoretic conjecture, constructs in polynomial time a 5/6-well-supported Nash equilibrium. However, the status of this conjecture is still unknown. In [6] it is also shown how to transform a $[0, 1]$-bimatrix game to a $\{0, 1\}$-bimatrix game of the same size, so that each ϵ-well supported Nash equilibrium of the resulting game is $(1 + \epsilon)/2$-well supported Nash equilibrium of the original game.

The work of Kontogiannis and Spirakis [11] provides a polynomial algorithm that computes a 1/2-well-supported Nash equilibrium for arbitrary win-lose games. The idea behind this algorithm is to split evenly the divergence from a zero sum game between the two players and then solve this zero sum game in polynomial time (using its direct connection to Linear Programming). The computed Nash equilibrium of the zero sum game considered is indeed proved to be also a 1/2-well-supported Nash equilibrium for the initial win-lose game. Therefore:

Theorem 6 ([11]) *For any win-lose bimatrix game, there is a polynomial time constructable profile that is a* 1/2*-well-supported Nash equilibrium of the game.*

In the same work, Kontogiannis and Spirakis [11] parametrize the above methodology in order to apply it to arbitrary bimatrix games. This new technique leads to a weaker φ-well-supported Nash equilibrium for win-lose games, where $\phi = (\sqrt{5} - 1)/2$ is the golden ratio. Nevertheless, this parametrized technique extends nicely to a technique for arbitrary bimatrix games, which assures a 0.658-well-supported Nash equilibrium in polynomial time:

Theorem 7 ([11]) *For any bimatrix game, a* $\left(\sqrt{11}/2 - 1\right)$*-well-supported Nash equilibrium is constructable in polynomial time.*

Two very new results improved the approximation status of ϵ- Nash Equilibria:

Theorem 8 ([2]) *There is a polynomial time algorithm, based on Linear Programming, that provides an 0.36392-Nash Equilibrium.*

The second result below is the best till now:

Theorem 9 ([17]) *There exists a polynomial time algorithm, based on the stationary points of a natural optimization problem, that provides an 0.3393-Nash Equilibrium.*

Kannan and Theobald [9] investigate a hierarchy of bimatrix games $\langle A, B \rangle$ which results from restricting the rank of the matrix $A + B$ to be of fixed rank at most k. They propose a new model of ϵ-approximation for games of rank k and, using results from quadratic optimization, show that approximate Nash equilibria of constant rank games can be computed deterministically in time polynomial in $1/\epsilon$. Moreover, [9] provides a randomized approximation algorithm for certain quadratic optimization problems, which yields a randomized approximation algorithm for the Nash equilibrium problem. This randomized algorithm has similar time complexity as the deterministic one, but it has the possibility of finding an exact solution in polynomial time if a conjecture is valid. Finally, they present a polynomial time algorithm for *relative approximation* (with respect to the payoffs in an equilibrium) provided that the matrix $A + B$ has a nonnegative decomposition.

Applications

Non-cooperative game theory and its main solution concept, i. e. the Nash equilibrium, have been extensively used to understand the phenomena observed when decision-makers interact and have been applied in many diverse academic fields, such as biology, economics, sociology and artificial intelligence. Since however the computation of a Nash equilibrium is in general PPAD-complete, it is important to provide efficient algorithms for approximating a Nash equilibrium; the algorithms discussed in this entry are a first step towards this direction.

Cross References

- ► Complexity of Bimatrix Nash Equilibria
- ► General Equilibrium
- ► Non-approximability of Bimatrix Nash Equilibria

Recommended Reading

1. Althöfer, I.: On sparse approximations to randomized strategies and convex combinations. Linear Algebr. Appl. **199**, 339–355 (1994)
2. Bosse, H., Byrka, J., Markakis, E.: New Algorithms for Approximate Nash Equilibria in Bimatrix Games. In: LNCS Proceedings of the 3rd International Workshop on Internet and Network Economics (WINE 2007), San Diego, 12–14 December 2007
3. Chen, X., Deng, X.: Settling the complexity of 2-player Nash-equilibrium. In: Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06). Berkeley, 21–24 October 2005
4. Chen, X., Deng, X., Teng, S.-H.: Computing Nash equilibria: Approximation and smoothed complexity. In: Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06), Berkeley, 21–24 October 2006
5. Daskalakis, C., Goldberg, P., Papadimitriou, C.: The complexity of computing a Nash equilibrium. In: Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC'06), pp. 71–78. Seattle, 21–23 May 2006
6. Daskalakis, C., Mehta, A., Papadimitriou, C.: A note on approximate Nash equilibria. In: Proceedings of the 2nd Workshop on Internet and Network Economics (WINE'06), pp. 297–306. Patras, 15–17 December 2006
7. Daskalakis, C., Mehta, A., Papadimitriou, C: Progress in approximate Nash equilibrium. In: Proceedings of the 8th ACM Conference on Electronic Commerce (EC07), San Diego, 11–15 June 2007
8. Daskalakis, C., Papadimitriou, C.: Three-player games are hard. In: Electronic Colloquium on Computational Complexity (ECCC) (2005)
9. Kannan, R., Theobald, T.: Games of fixed rank: A hierarchy of bimatrix games. In: Proceedings of the ACM-SIAM Symposium on Discrete Algorithms, New Orleans, 7–9 January 2007
10. Kontogiannis, S., Panagopoulou, P.N., Spirakis, P.G.: Polynomial algorithms for approximating Nash equilibria of bimatrix games. In: Proceedings of the 2nd Workshop on Internet and Network Economics (WINE'06), pp. 286–296. Patras, 15–17 December 2006
11. Kontogiannis, S., Spirakis, P.G.: Efficient Algorithms for Constant Well Supported Approximate Equilibria in Bimatrix Games. In: Proceedings of the 34th International Colloquium on Automata, Languages and Programming (ICALP'07, Track A: Algorithms and Complexity), Wroclaw, 9–13 July 2007
12. Lemke, C.E., Howson, J.T.: Equilibrium points of bimatrix games. J. Soc. Indust. Appl. Math. **12**, 413–423 (1964)
13. Lipton, R.J., Markakis, E., Mehta, A.: Playing large games using simple startegies. In: Proceedings of the 4th ACM Conference on Electronic Commerce (EC'03), pp. 36–41. San Diego, 9–13 June 2003
14. Nash, J.: Noncooperative games. Ann. Math. **54**, 289–295 (1951)
15. Papadimitriou, C.H.: On inefficient proofs of existence and complexity classes. In: Proceedings of the 4th Czechoslovakian Symposium on Combinatorics 1990, Prachatice (1991)
16. Savani, R., von Stengel, B.: Exponentially many steps for finding a nash equilibrium in a bimatrix game. In: Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS'04), pp. 258–267. Rome, 17–19 October 2004
17. Tsaknakis, H., Spirakis, P.: An Optimization Approach for Approximate Nash Equilibria. In: LNCS Proceedings of the 3rd International Workshop on Internet and Network Economics (WINE 2007), also in the Electronic Colloquium on Computational Complexity, (ECCC), TR07-067 (Revision), San Diego, 12–14 December 2007

18. von Neumann, J., Morgenstern, O.: Theory of Games and Economic Behavior. Princeton University Press, Princeton, NJ (1944)

Approximation Schemes for Bin Packing
1982; Karmarker, Karp

NIKHIL BANSAL
IBM Research, IBM, Yorktown Heights, NY, USA

Keywords and Synonyms

Cutting stock problem

Problem Definition

In the bin packing problem, the input consists of a collection of items specified by their sizes. There are also identical bins, which without loss of generality can be assumed to be of size 1, and the goal is to pack these items using the minimum possible number of bins.

Bin packing is a classic optimization problem, and hundreds of its variants have been defined and studied under various settings such as average case analysis, worst-case offline analysis, and worst-case online analysis. This note considers the most basic variant mentioned above under the offline model where all the items are given in advance. The problem is easily seen to be NP-hard by a reduction from the partition problem. In fact, this reduction implies that unless P = NP, it impossible to determine in polynomial time whether the items can be packed into two bins or whether they need three bins.

Notations

The input to the bin packing problem is a set of n items I specified by their sizes $s_1, \dots, s_n$, where each s_i is a real number in the range $(0, 1]$. A subset of items $S \subseteq I$ can be packed feasibly in a bin if the total size of items in S is at most 1. The goal is to pack all items in I into the minimum number of bins. Let $\text{OPT}(I)$ denote the value of the optimum solution and $\text{Size}(I)$ the total size of all items in I. Clearly, $\text{OPT}(I) \geq \lceil \text{Size}(I) \rceil$.

Strictly speaking, the problem does not admit a polynomial-time algorithm with an approximation guarantee better than 3/2. Interestingly, however, this does not rule out an algorithm that requires, say, $\text{OPT}(I) + 1$ bins (unlike other optimization problems, making several copies of a small hard instance to obtain a larger hard instance does not work for bin packing). It is more meaningful to consider approximation guarantees in an asymptotic sense. An algorithm is called an asymptotic ρ approximation if the number of bins required by it is $\rho \cdot \text{OPT}(I) + O(1)$.

Key Results

During the 1960s and 1970s several algorithms with constant factor asymptotic and absolute approximation guarantees and very efficient running times were designed (see [1] for a survey). A breakthrough was achieved in 1981 by de la Vega and Lueker [3], who gave the first polynomial-time asymptotic approximation scheme.

Theorem 1 ([3]) *Given any arbitrary parameter $\epsilon > 0$, there is an algorithm that uses $(1 + \epsilon)OPT(I) + O(1)$ bins to pack I. The running time of this algorithm is $O(n \log n) + (1/\epsilon)^{O(1/\epsilon)}$.*

The main insight of de la Vega and Lueker [3] was to give a technique for approximating the original instance by a simpler instance where large items have only $O(1)$ distinct sizes. Their idea was simple. First, it suffices to restrict attention to large items, say, with size greater than ε. These can be called I_b. Given an (almost) optimum packing of I_b, consider the solution obtained by greedily filling up the bins with remaining small items, opening new bins only if needed. Indeed, if no new bins are needed, then the solution is still almost optimum since the packing for I_b was almost optimum. If additional bins are needed, then each bin, except possibly one, must be filled to an extent $(1 - \epsilon)$, which gives a packing using $\text{Size}(I)/(1 - \epsilon) + 1 \leq \text{OPT}(I)/(1 - \epsilon) + 1$ bins. So it suffices to focus on solving I_b almost optimally. To do this, the authors show how to obtain another instance I' with the following properties. First, I' has only $O(1/\epsilon^2)$ distinct sizes, and second, I' is an approximation of I_b in the sense that $\text{OPT}(I_b) \geq \text{OPT}(I')$ and, moreover, any solution of I' implies another solution of I_b using $O(\epsilon \cdot \text{OPT}(I))$ additional bins. As I' has only $1/\epsilon^2$ distinct item sizes, and any bin can obtain at most $1/\epsilon$ such items, there are at most $O(1/\epsilon^2)^{1/\epsilon}$ ways to pack a bin. Thus, I' can be solved optimally by exhaustive enumeration (or more efficiently using an integer programming formulation described below).

Later, Karmarkar and Karp [4] proved a substantially stronger guarantee.

Theorem 2 ([4]) *Given an instance I, there is an algorithm that produces a packing of I using $OPT(I) + O(\log^2 OPT(I))$ bins. The running time of this algorithm is $O(n^8)$.*

Observe that this guarantee is significantly stronger than that of [3] as the additive term is $O(\log^2 \text{OPT})$ as opposed to $O(\epsilon \cdot \text{OPT})$. Their algorithm also uses the ideas of reducing the number of distinct item sizes and ignoring small items, but in a much more refined way. In particular, instead of obtaining a rounded instance in a single step, their algorithm consists of a logarithmic number of steps where in each step they round the instance "mildly" and then solve it partially.

The starting point is an exponentially large linear programming (LP) relaxation of the problem commonly referred to as the configuration LP. Here there is a variable x_S corresponding to each subset of items S that can be packed feasibly in a bin. The objective is to minimize $\sum_S x_S$ subject to the constraint that for each item i, the sum of x_S over all subsets S that contain i is at least 1. Clearly, this is a relaxation as setting $x_S = 1$ for each set S corresponding to a bin in the optimum solution is a feasible integral solution to the LP. Even though this formulation has exponential size, the separation problem for the dual is a knapsack problem, and hence the LP can be solved in polynomial time to any accuracy (in particular within an accuracy of 1) using the ellipsoid method. Such a solution is called a fractional packing. Observe that if there are n_i items each of size exactly s_i, then the constraints corresponding to i can be "combined" to obtain the following LP:

$$\begin{aligned} &\min \sum_S x_S && \\ \text{s.t.}\quad &\sum_S a_{S,i} x_S \geq n_i && \forall \text{ item sizes } i \\ &x_S \geq 0 && \forall \text{ feasible sets } S. \end{aligned}$$

Here $a_{S,i}$ is the number of items of size s_i in the feasible S. Let $q(I)$ denote the number of distinct sizes in I. The number of nontrivial constraints in LP is equal to $q(I)$, which implies that there is a basic optimal solution to this LP that has only $q(I)$ variables set nonintegrally. Karmarkar and Karp exploit this observation in a very clever way. The following lemma describes the main idea.

Lemma 3 *Given any instance J, suppose there is an algorithmic rounding procedure to obtain another instance J' such that J' has Size$(J)/2$ distinct item sizes and J and J' are related in the following sense: given any fractional packing of J using ℓ bins gives a fractional packing of J' with at most ℓ bins, and given any packing of J' using ℓ' bins gives a packing of J using $\ell' + c$ bins, where c is some fixed parameter. Then J can be packed using $OPT(J) + c \cdot \log(OPT(J))$ bins.*

Proof Let $I_0 = I$ and let I_1 be the instance obtained by applying the rounding procedure to I_0. By the property of the rounding procedure, $\mathrm{OPT}(I) \leq \mathrm{OPT}(I_1) + c$ and $\mathrm{LP}(I_1) \leq \mathrm{LP}(I)$. As I_1 has Size$(I_0)/2$ distinct sizes, the LP solution for I_1 has at most Size$(I_0)/2$ fractionally set variables. Remove the items packed integrally in the LP solution and consider the residual instance I_1'. Note that Size$(I_1') \leq$ Size$(I_0)/2$. Now, again apply the rounding procedure to I_1' to obtain I_2 and solve the LP for I_2. Again, this solution has at most Size$(I_1')/2 \leq$ Size$(I_0)/4$ fractionally set variables, and $\mathrm{OPT}(I_1') \leq \mathrm{OPT}(I_2) + c$ and $\mathrm{LP}(I_2) \leq \mathrm{LP}(I_1')$. The above process is repeated for a few steps. At each step, the size of the residual instance decreases by a factor of at least two, and the number of bins required to pack I_0 increases by additive c. After $\log(\text{Size}(I_0))$ ($\approx \log(\mathrm{OPT}(I))$) steps, the residual instance has size $O(1)$ and can be packed into $O(1)$ additional bins. □

It remains to describe the rounding procedure. Consider the items in nondecreasing order $s_1 \geq s_2 \geq \ldots \geq s_n$ and group them as follows. Add items to current group until its size first exceeds 2. At this point close the group and start a new group. Let $G_1, \ldots, G_k$ denote the groups formed and let $n_i = |G_i|$, setting $n_0 = 0$ for convenience. Define I' as the instance obtained by rounding the size of n_{i-1} largest items in G_i to the size of the largest item in G_i for $i = 1, \ldots, k$. The procedure satisfies the properties of Lemma 3 with $c = O(\log n_k)$ (left as an exercise to the reader). To prove Theorem 2, it suffices to show that $n_k = O(\text{Size}(I))$. This is done easily by ignoring all items smaller than $1/\text{Size}(I)$ and filling them in only in the end (as in the algorithm of de la Vega and Lueker).

In the case when the item sizes are not too small, the following corollary is obtained.

Corollary 1 *If all the item sizes are at least δ, it is easily seen that $c = O(\log 1/\delta)$, and the above algorithm implies a guarantee of $OPT + O(\log(1/\delta) \cdot \log OPT)$, which is $OPT + O(\log OPT)$ if δ is a constant.*

Applications

The bin packing problem is directly motivated from practice and has many natural applications such as packing items into boxes subject to weight constraints, packing files into CDs, packing television commercials into station breaks, and so on. It is widely studied in operations research and computer science. Other applications include the so-called cutting-stock problems where some material such as cloth or lumber is given in blocks of standard size from which items of certain specified size must be cut. Several variations of bin packing, such as generalizations to higher dimensions, imposing additional constraints on the algorithm and different optimization criteria, have also been extensively studied. The reader is referred to [1,2] for excellent surveys.

Open Problems

Except for the NP-hardness, no other hardness results are known and it is possible that a polynomial-time algorithm with guarantee OPT + 1 exists for the problem. Resolving this is a key open question. A promising approach seems to be via the configuration LP (considered above). In fact, no instance is known for which the additive gap between the optimum configuration LP solution and the optimum integral solution is more than 1. It would be very interesting to design an instance that has an additive integrality gap of two or more.

The OPT + $O(\log^2 \text{OPT})$ guarantee of Karmarkar and Karp has been the best known result for the last 25 years, and any improvement to this would be an extremely interesting result by itself.

Cross References

▶ Bin Packing
▶ Knapsack

Recommended Reading

1. Coffman, E.G., Garey, M.R., Johnson, D.S.: Approximation algorithms for bin packing: a survey. In: Hochbaum, D. (ed.) Approximation Algorithms for NP-hard Problems, pp. 46–93. PWS, Boston (1996)
2. Csirik, J., Woeginger, G.: On-line packing and covering problems. In: Fiat, A., Woeginger, G. (eds.) Online Algorithms: The State of the Art. LNCS, vol. 1442, pp. 147–177. Springer, Berlin (1998)
3. Fernandez de la Vega, W., Lueker, G.: Bin packing can be solved within $1 + \varepsilon$ in linear time. Combinatorica **1**, 349–355 (1981)
4. Karmarkar, N., Karp, R.M.: An efficient approximation scheme for the one-dimensional bin-packing problem. In: Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science (FOCS), 1982, pp. 312–320

Approximation Schemes for Planar Graph Problems

1983; Baker
1994; Baker

Erik D. Demaine[1], MohammadTaghi Hajiaghayi[2]
[1] Computer Science and Artifical Intelligence Laboratory, MIT, Cambridge, MA, USA
[2] Department of Computer Science, University of Pittsburgh, Pittsburgh, PA, USA

Keywords and Synonyms

Approximation algorithms in planar graphs; Baker's approach; Lipton–Tarjan approach

Problem Definition

Many NP-hard graph problems become easier to approximate on planar graphs and their generalizations. (A graph is *planar* if it can be drawn in the plane (or the sphere) without crossings. For definitions of other related graph classes, see the entry on ▶ bidimensionality (2004; Demaine, Fomin, Hajiaghayi, Thilikos).) For example, *maximum independent set* asks to find a maximum subset of vertices in a graph that induce no edges. This problem is inapproximable in general graphs within a factor of $n^{1-\epsilon}$ for any $\epsilon > 0$ unless NP = ZPP (and inapproximable within $n^{1/2-\epsilon}$ unless P = NP), while for planar graphs there is a 4-approximation (or simple 5-approximation) by taking the largest color class in a vertex 4-coloring (or 5-coloring). Another is *minimum dominating set*, where the goal is to find a minimum subset of vertices such that every vertex is either in or adjacent to the subset. This problem is inapproximable in general graphs within $\epsilon \log n$ for some $\epsilon > 0$ unless P = NP, but as we will see, for planar graphs the problem admits a *polynomial-time approximation scheme* (PTAS): a collection of $(1 + \epsilon)$-approximation algorithms for all $\epsilon > 0$.

There are two main general approaches to designing PTASs for problems on planar graphs and their generalizations: the separator approach and the Baker approach.

Lipton and Tarjan [15,16] introduced the first approach, which is based on planar separators. The first step in this approach is to find a separator of $O(\sqrt{n})$ vertices or edges, where n is the size of the graph, whose removal splits the graph into two or more pieces each of which is a constant fraction smaller than the original graph. Then recurse in each piece, building a recursion tree of separators, and stop when the pieces have some constant size such as $1/\epsilon$. The problem can be solved on these pieces by brute force, and then it remains to combine the solutions up the recursion tree. The induced error can often be bounded in terms of the total size of all separators, which in turn can be bounded by $\epsilon\, n$. If the optimal solution is at least some constant factor times n, this approach often leads to a PTAS.

There are two limitations to this planar-separator approach. First, it requires that the optimal solution be at least some constant factor times n; otherwise, the cost incurred by the separators can be far larger than the desired optimal solution. Such a bound is possible in some problems after some graph pruning (linear kernelization), e. g., independent set, vertex cover, and forms of the traveling salesman problem. But, for example, Grohe [12] states that the dominating set is a problem "to which the technique based on the separator theorem does not apply." Second, the approximation algorithms resulting from planar separators are often impractical because of large constant factors. For example, to achieve an approximation ratio of just 2, the base case requires exhaustive solution of graphs of up to $2^{2^{400}}$ vertices.

Baker [1] introduced her approach to address the second limitation, but it also addresses the first limitation to a certain extent. This approach is based on decomposition into overlapping subgraphs of bounded outerplanarity, as described in the next section.

Key Results

Baker's original result [1] is a PTAS for a maximum independent set (as defined above) on planar graphs, as well as the following list of problems on planar graphs: maximum tile salvage, partition into triangles, maximum H-matching, minimum vertex cover, minimum dominating set, and minimum edge-dominating set.

Baker's approach starts with a planar embedding of the planar graph. Then it divides vertices into *layers* by iteratively removing vertices on the outer face of the graph: layer j consists of the vertices removed at the jth iteration. If one now removes the layers congruent to i modulo k, for any choice of i, the graph separates into connected components each with at most k consecutive layers, and hence the graph becomes k-outerplanar. Many NP-complete problems can be solved on k-outerplanar graphs for fixed k using dynamic programming (in particular, such graphs have bounded treewidth). Baker's approximation algorithm computes these optimal solutions for each choice i of the congruence class of layers to remove and returns the best solution among these k solutions. The key argument for maximization problems considers the optimal solution to the full graph and argues that the removal of one of the k congruence classes of layers must remove at most a $1/k$ fraction of the optimal solution, so the returned solution must be within a $1 + 1/k$ factor of optimal. A more delicate argument handles minimization problems as well. For many problems, such as maximum independent set, minimum dominating set, and minimum vertex cover, Baker's approach obtains a $(1 + \epsilon)$-approximation algorithms with a running time of $2^{O(1/\epsilon)} n^{O(1)}$ on planar graphs.

Eppstein [10] generalized Baker's approach to a broader class of graphs called graphs of *bounded local treewidth*, i. e., where the treewidth of the subgraph induced by the set of vertices at a distance of at most r from any vertex is bounded above by some function $f(r)$ independent of n. The main differences in Eppstein's approach are replacing the concept of bounded outerplanarity with the concept of bounded treewidth, where dynamic programming can still solve many problems, and labeling layers according to a simple breadth-first search. This approach has led to PTASs for hereditary maximization problems such as maximum independent set and maximum clique, maximum triangle matching, maximum H-matching, maximum tile salvage, minimum vertex cover, minimum dominating set, minimum edge-dominating set, minimum color sum, and subgraph isomorphism for a fixed pattern [6,8,10]. Frick and Grohe [11] also developed a general framework for deciding any property expressible in first-order logic in graphs of bounded local treewidth.

The foundation of these results is Eppstein's characterization of minor-closed families of graphs with bounded local treewidth [10]. Specifically, he showed that a minor-closed family has bounded local treewidth if and only if it excludes some *apex graph*, a graph with a vertex whose removal leaves a planar graph. Unfortunately, the initial proof of this result brought Eppstein's approach back to the realm of impracticality, because his bound on local treewidth in a general apex-minor-free graph is doubly exponential in r: $2^{2^{O(r)}}$. Fortunately, this bound could be improved to $2^{O(r)}$ [3] and even the optimal $O(r)$ [4]. The latter bound restores Baker's $2^{O(1/\epsilon)} n^{O(1)}$ running time for $(1 + \epsilon)$-approximation algorithms, now for all apex-minor-free graphs.

Another way to view the necessary decomposition of Baker's and Eppstein's approaches is that the vertices or edges of the graph can be split into any number k of pieces such that deleting any one of the pieces results in a graph of bounded treewidth (where the bound depends on k). Such decompositions in fact exist for arbitrary graphs excluding any fixed minor H [9], and they can be found in polynomial time [6]. This approach generalizes the Baker–Eppstein PTASs described above to handle general H-minor-free graphs.

This decomposition approach is effectively limited to *deletion-closed* problems, whose optimal solution only improves when deleting edges or vertices from the graph. Another decomposition approach targets *contraction-closed* problems, whose optimal solution only improves when contracting edges. These problems include classic problems such as dominating set and its variations, the traveling salesman problem, subset TSP, minimum Steiner tree, and minimum-weight c-edge-connected submultigraph. PTASs have been obtained for these problems in planar graphs [2,13,14] and in bounded-genus graphs [7] by showing that the edges can be decomposed into any number k of pieces such that contracting any one piece results in a bounded-treewidth graph (where the bound depends on k).

Applications

Most applications of Baker's approach have been limited to optimization problems arising from "local" properties (such as those definable in first-order logic). Intuitively, such local properties can be decided by locally checking every constant-size neighborhood. In [5], Baker's approach is generalized to obtain PTASs for nonlocal problems, in particular, connected dominating set. This generalization requires the use of two different techniques. The first technique is to use an ε-fraction of a constant-factor (or even logarithmic-factor) approximation to the problem as a "backbone" for achieving the needed nonlocal property. The second technique is to use subproblems that overlap by $\Theta(\log n)$ layers instead of the usual $\Theta(1)$ in Baker's approach.

Despite this advance in applying Baker's approach to more general problems, the planar-separator approach can still handle some different problems. Recall, though, that the planar-separator approach was limited to problems in which the optimal solution is at least some constant factor times n. This limitation has been overcome for a wide range of problems [5], in particular obtaining a PTAS for feedback vertex set, to which neither Baker's approach nor the planar-separator approach could previously apply. This result is based on evenly dividing the optimum solution instead of the whole graph, using a relation between treewidth and the optimal solution value to bound the treewidth of the graph, and thus obtaining an $O(\sqrt{\mathrm{OPT}})$ separator instead of an $O(\sqrt{n})$ separator. The $O(\sqrt{\mathrm{OPT}})$ bound on treewidth follows from the bidimensionality theory described in the entry on ▶ bidimensionality (2004; Demaine, Fomin, Hajiaghayi, Thilikos). We can divide the optimum solution into roughly even pieces, without knowing the optimum solution, by using existing constant-factor (or even logarithmic-factor) approximations for the problem. At the base of the recursion, pieces no longer have bounded size but do have bounded treewidth, so fast fixed-parameter algorithms can be used to construct optimal solutions.

Open Problems

An intriguing direction for future research is to build a general theory for PTASs of subset problems. Although PTASs for subset TSP and Steiner tree have recently been obtained for planar graphs [2,14], there remain several open problems of this kind, such as subset feedback vertex set.

Another instructive problem is to understand the extent to which Baker's approach can be applied to nonlocal problems. Again there is an example of how to modify the approach to handle the nonlocal problem of connected dominating set [5], but for example the only known PTAS for feedback vertex set in planar graphs follows the separator approach.

Cross References

▶ Bidimensionality
▶ Separators in Graphs
▶ Treewidth of Graphs

Recommended Reading

1. Baker, B.S.: Approximation algorithms for NP-complete problems on planar graphs. J. Assoc. Comput. Mach. **41**(1), 153–180 (1994)
2. Borradaile, G., Kenyon-Mathieu, C., Klein, P.N.: A polynomial-time approximation scheme for Steiner tree in planar graphs. In: Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms, 2007
3. Demaine, E.D., Hajiaghayi, M.: Diameter and treewidth in minor-closed graph families, revisited. Algorithmica **40**(3), 211–215 (2004)
4. Demaine, E.D., Hajiaghayi, M.: Equivalence of local treewidth and linear local treewidth and its algorithmic applications. In: Proceedings of the 15th ACM-SIAM Symposium on Discrete Algorithms (SODA'04), January 2004, pp. 833–842
5. Demaine, E.D., Hajiaghayi, M.: Bidimensionality: new connections between FPT algorithms and PTASs. In: Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2005), Vancouver, January 2005, pp. 590–601
6. Demaine, E.D., Hajiaghayi, M., Kawarabayashi, K.-I.: Algorithmic graph minor theory: Decomposition, approximation, and coloring. In: Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science, Pittsburgh, October 2005, pp. 637–646
7. Demaine, E.D., Hajiaghayi, M., Mohar, B.: Approximation algorithms via contraction decomposition. In: Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms, New Orleans, 7–9 January 2007, pp. 278–287
8. Demaine, E.D., Hajiaghayi, M., Nishimura, N., Ragde, P., Thilikos, D.M.: Approximation algorithms for classes of graphs excluding single-crossing graphs as minors. J. Comput. Syst. Sci. **69**(2), 166–195 (2004)
9. DeVos, M., Ding, G., Oporowski, B., Sanders, D.P., Reed, B., Seymour, P., Vertigan, D.: Excluding any graph as a minor allows a low tree-width 2-coloring. J. Comb. Theory Ser. B **91**(1), 25–41 (2004)
10. Eppstein, D.: Diameter and treewidth in minor-closed graph families. Algorithmica **27**(3–4), 275–291 (2000)
11. Frick, M., Grohe, M.: Deciding first-order properties of locally tree-decomposable structures. J. ACM **48**(6), 1184–1206 (2001)
12. Grohe, M.: Local tree-width, excluded minors, and approximation algorithms. Combinatorica **23**(4), 613–632 (2003)
13. Klein, P.N.: A linear-time approximation scheme for TSP for planar weighted graphs. In: Proceedings of the 46th IEEE Symposium on Foundations of Computer Science, 2005, pp. 146–155
14. Klein, P.N.: A subset spanner for planar graphs, with application to subset TSP. In: Proceedings of the 38th ACM Symposium on Theory of Computing, 2006, pp. 749–756

15. Lipton, R.J., Tarjan, R.E.: A separator theorem for planar graphs. SIAM J. Appl. Math. **36**(2), 177–189 (1979)
16. Lipton, R.J., Tarjan, R.E.: Applications of a planar separator theorem. SIAM J. Comput. **9**(3), 615–627 (1980)

Arbitrage in Frictional Foreign Exchange Market
2003; Cai, Deng

Mao-cheng Cai[1], Xiaotie Deng[2]
[1] Institute of Systems Science, Chinese Academy of Sciences, Beijing, China
[2] Department of Computer Science, City University of Hong Kong, Hong Kong, China

Problem Definition

The simultaneous purchase and sale of the same securities, commodities, or foreign exchange in order to profit from a differential in the price. This usually takes place on different exchanges or marketplaces. Also known as a "Riskless profit".

Arbitrage is, arguably, the most fundamental concept in finance. It is a state of the variables of financial instruments such that a riskless profit can be made, which is generally believed not in existence. The economist's argument for its non-existence is that active investment agents will exploit any arbitrage opportunity in a financial market and thus will deplete it as soon as it may arise. Naturally, the speed at which such an arbitrage opportunity can be located and be taken advantage of is important for the profit-seeking investigators, which falls in the realm of analysis of algorithms and computational complexity.

The identification of arbitrage states is, at frictionless foreign exchange market (a theoretical trading environment where all costs and restraints associated with transactions are non-existent), not difficult at all and can be reduced to existence of arbitrage on three currencies (see [11]). In reality, friction does exist. Because of friction, it is possible that there exist arbitrage opportunities in the market but difficult to find it and to exploit it to eliminate it. Experimental results in foreign exchange markets showed that arbitrage does exist in reality. Examination of data from ten markets over a twelve day period by Mavrides [11] revealed that a significant arbitrage opportunity exists. Some opportunities were observed to be persistent for a long time. The problem become worse at forward and futures markets (in which futures contracts in commodities are traded) coupled with covered interest rates, as observed by Abeysekera and Turtle [1], and Clinton [4]. An obvious interpretation is that the arbitrage opportunity was not immediately identified because of information asymmetry in the market. However, that is not the only factor. Both the time necessary to collect the market information (so that an arbitrage opportunity would be identified) and the time people (or computer programs) need to find the arbitrage transactions are important factors for eliminating arbitrage opportunities.

The computational complexity in identifying arbitrage, the level in difficulty measured by arithmetic operations, is different in different models of exchange systems. Therefore, to approximate an ideal exchange market, models with lower complexities should be preferred to those with higher complexities.

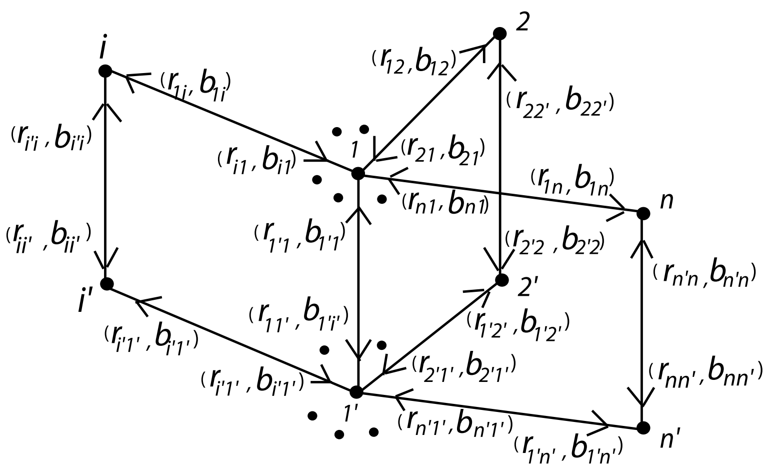
To model an exchange system, consider n foreign currencies: $N = \{1, 2, \dots, n\}$. For each ordered pair (i, j), one may change one unit of currency i to r_{ij} units of currency j. Rate r_{ij} is the *exchange rate* from i to j. In an ideal market, the exchange rate holds for any amount that is exchanged. An *arbitrage opportunity* is a set of exchanges between pairs of currencies such that the net balance for each involved currency is non-negative and there is at least one currency for which the net balance is positive. Under ideal market conditions, there is no arbitrage if and only if there is no arbitrage among any three currencies (see [11]).

Various types of *friction* can be easily modeled in such a system. Bid-offer spread may be expressed in the present mathematical format as $r_{ij} r_{ji} < 1$ for some $i, j \in N$. In addition, usually the traded amount is required to be in multiples of a fixed integer amount, hundreds, thousands or millions. Moreover, different traders may bid or offer at different rates, and each for a limited amount. A more general model to describe these market *imperfections* will include, for pairs $i \neq j \in N$, l_{ij} different rates r^k_{ij} of exchanges from currency i to j up to b^k_{ij} units of currency i, $k = 1, \dots, l_{ij}$, where l_{ij} is the number of different exchange rates from currency i to j.

A currency exchange market can be represented by a digraph $G = (V, E)$ with vertex set V and arc set E such that each vertex $i \in V$ represents currency i and each arc $a^k_{ij} \in E$ represents the currency exchange relation from i to j with rate r^k_{ij} and bound b^k_{ij}. Note that parallel arcs may occur for different exchange rates. Such a digraph is called an exchange digraph. Let $x = (x^k_{ij})$ denote a currency exchange vector.

Problem 1 *The existence of arbitrage in a frictional exchange market can be formulated as follows.*

$$\sum_{j \neq i} \sum_{k=1}^{l_{ji}} \lfloor r^k_{ji} x^k_{ji} \rfloor - \sum_{j \neq i} \sum_{k=1}^{l_{ij}} x^k_{ij} \geq 0, \quad i = 1, \dots, n, \quad (1)$$



Arbitrage in Frictional Foreign Exchange Market, Figure 1
Digraph G_1

at least one strict inequality holds

$$0 \le x_{ij}^k \le b_{ij}^k,\ 1 \le k \le l_{ij},\ 1 \le i \ne j \le n\,, \qquad (2)$$

$$x_{ij}^k \text{ is integer},\ 1 \le k \le l_{ij},\ 1 \le i \ne j \le n. \qquad (3)$$

Note that the first term in the right hand side of (1) is the revenue at currency i by selling other currencies and the second term is the expense at currency i by buying other currencies.

The corresponding optimization problem is

Problem 2 *The maximum arbitrage problem in a frictional foreign exchange market with bid-ask spreads, bound and integrality constraints is the following integer linear programming (P):*

$$\text{maximize} \sum_{i=1}^{n} w_i \sum_{j \ne i} \left(\sum_{k=1}^{l_{ji}} \lfloor r_{ji}^k x_{ji}^k \rfloor - \sum_{k=1}^{l_{ij}} x_{ij}^k \right)$$

subject to

$$\sum_{j \ne i} \left(\sum_{k=1}^{l_{ji}} \lfloor r_{ji}^k x_{ji}^k \rfloor - \sum_{k=1}^{l_{ij}} x_{ij}^k \right) \ge 0\,, \quad i = 1, \dots, n\,, \quad (4)$$

$$0 \le x_{ij}^k \le b_{ij}^k\,, \quad 1 \le k \le l_{ij}\,, \quad 1 \le i \ne j \le n\,, \quad (5)$$

$$x_{ij}^k \text{ is integer}\,, \quad 1 \le k \le l_{ij}\,, \quad 1 \le i \ne j \le n\,, \quad (6)$$

where $w_i \ge 0$ is a given weight for currency i, $i = 1, 2, \dots, n$, with at least one $w_i > 0$.

Finally consider another

Problem 3 *In order to eliminate arbitrage, how many transactions and arcs in a exchange digraph have to be used for the currency exchange system?*

Key Results

A decision problem is called nondeterministic polynomial (*NP* for short) if its solution (if one exists) can be guessed and verified in polynomial time; nondeterministic means that no particular rule is followed to make the guess. If a problem is *NP* and all other *NP* problems are polynomial-time reducible to it, the problem is *NP*-complete. And a problem is called *NP*-hard if every other problem in NP is polynomial-time reducible to it.

Theorem 1 *It is NP-complete to determine whether there exists arbitrage in a frictional foreign exchange market with bid-ask spreads, bound and integrality constraints even if all $l_{ij} = 1$.*

Then a further inapproximability result is obtained.

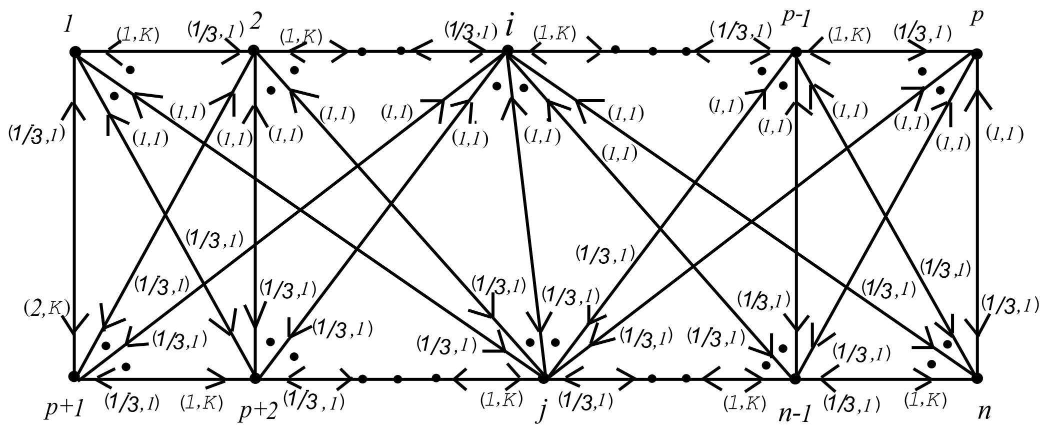
Theorem 2 *There exists fixed $\epsilon > 0$ such that approximating (P) within a factor of n^ϵ is NP-hard even for any of the following two special cases:*

(P_1) *all $l_{ij} = 1$ and $w_i = 1$.*
(P_2) *all $l_{ij} = 1$ and all but one $w_i = 0$.*

Now consider two polynomially solvable special cases when the number of currencies is constant or the exchange digraph is star-shaped (a digraph is star-shaped if all arcs have a common vertex).

Theorem 3 *There are polynomial time algorithms for (P) when the number of currencies is constant.*

Theorem 4 *It is polynomially solvable to find the maximum revenue at the center currency of arbitrage in a frictional foreign exchange market with bid-ask spread, bound*



Arbitrage in Frictional Foreign Exchange Market, Figure 2
Digraph G_2

and integrality constraints when the exchange digraph is star-shaped.

However, if the exchange digraph is the coalescence of a star-shaped exchange digraph and its copy, shown by Digraph G_1, then the problem becomes *NP*-complete.

Theorem 5 *It is NP-complete to decide whether there exists arbitrage in a frictional foreign exchange market with bid-ask spreads, bound and integrality constraints even if its exchange digraph is coalescent.*

Finally an answer to Problem 3 is as follows.

Theorem 6 *There is an exchange digraph of order n such that at least $\lfloor n/2 \rfloor \lceil n/2 \rceil - 1$ transactions and at least $n^2/4 + n - 3$ arcs are in need to bring the system back to non-arbitrage states.*

For instance, consider the currency exchange market corresponding to digraph $G_2 = (V, E)$, where the number of currencies is $n = |V|$, $p = \lfloor n/2 \rfloor$ and $K = n^2$.

Set

$$\begin{aligned} C = \{a_{ij} \in E \,|\, 1 \le i \le p, p+1 \le j \le n\} \\ \cup \{a_{1(p+1)}\} \setminus \{a_{(p+1)1}\} \cup \{a_{i(i-1)} \,|\, 2 \le i \le p\} \\ \cup \{a_{i(i+1)} \,|\, p+1 \le i \le n-1\} \,. \end{aligned}$$

Then $|C| = \lfloor n/2 \rfloor \lceil n/2 \rceil + n - 2 = |E|/2 > n^2/4 + n - 3$. It follows easily from the rates and bounds that each arc in C has to be used to eliminate arbitrage. And $\lfloor n/2 \rfloor \lceil n/2 \rceil - 1$ transactions corresponding to $\{a_{ij} \in E \,|\, 1 \le i \le p, p + 1 \le j \le n\} \setminus \{a_{(p+1)1}\}$ are in need to bring the system back to non-arbitrage states.

Applications

The present results show that different foreign exchange systems exhibit quite different computational complexities. They may shed new light on how monetary system models are adopted and evolved in reality. In addition, it provides with a computational complexity point of view to the understanding of the now fast growing Internet electronic exchange markets.

Open Problems

The dynamic models involving in both spot markets (in which goods are sold for cash and delivered immediately) and futures markets are the most interesting ones. To develop good approximation algorithms for such general models would be important. In addition, it is also important to identify special market models for which polynomial time algorithms are possible even with future markets. Another interesting paradox in this line of study is why friction constraints that make arbitrage difficult are not always eliminated in reality.

Cross References

► General Equilibrium

Recommended Reading

1. Abeysekera, S.P., Turtle H.J.: Long-run relations in exchange markets: a test of covered interest parity. J. Financial Res. **18**(4), 431–447 (1995)
2. Ausiello, G., Crescenzi, P., Gambosi, G., Kann, V., Marchetti-Spaccamela, A., Protasi, M.: Complexity and approximation: combinatorial optimization problems and their approximability properties. Springer, Berlin (1999)

3. Cai, M., Deng, X.: Approximation and computation of arbitrage in frictional foreign exchange market. Electron. Notes Theor. Comput. Sci. **78**, 1–10(2003)
4. Clinton, K.: Transactions costs and covered interest arbitrage: theory and evidence. J. Politcal Econ. **96**(2), 358–370 (1988)
5. Deng, X., Li, Z.F., Wang, S.: Computational complexity of arbitrage in frictional security market. Int. J. Found. Comput. Sci. **13**(5), 681–684 (2002)
6. Deng, X., Papadimitriou, C.: On the complexity of cooperative game solution concepts. Math. Oper. Res. **19**(2), 257–266 (1994)
7. Deng, X., Papadimitriou, C., Safra, S.: On the complexity of price equilibria. J. Comput. System Sci. **67**(2), 311–324 (2003)
8. Garey, M.R., Johnson, D.S.: Computers and intractability: a guide of the theory of *NP*-completeness. Freeman, San Francisco (1979)
9. Jones, C.K.: A network model for foreign exchange arbitrage, hedging and speculation. Int. J. Theor. Appl. Finance **4**(6), 837–852 (2001)
10. Lenstra Jr., H.W.: Integer programming with a fixed number of variables. Math. Oper. Res. **8**(4), 538–548 (1983)
11. Mavrides, M.: Triangular arbitrage in the foreign exchange market – inefficiencies, technology and investment opportunities. Quorum Books, London (1992)
12. Megiddo, N.: Computational complexity and the game theory approach to cost allocation for a tree. Math. Oper. Res. **3**, 189–196 (1978)
13. Mundell, R.A.: Currency areas, exchange rate systems, and international monetary reform, paper delivered at Universidad del CEMA, Buenos Aires, Argentina. http://www.robertmundell.net/pdf/Currency (2000). Accessed 17 Apr 2000
14. Mundell, R.A.: Gold Would Serve into the 21st Century. Wall Street Journal, 30 September 1981, pp. 33
15. Zhang, S., Xu, C., Deng, X.: Dynamic arbitrage-free asset pricing with proportional transaction costs. Math. Finance **12**(1), 89–97 (2002)

Arithmetic Coding for Data Compression

1994; Howard, Vitter

PAUL G. HOWARD[1], JEFFREY SCOTT VITTER[2]
[1] Microway, Inc., Plymouth, MA, USA
[2] Department of Computer Science, Purdue University, West Lafayette, IN, USA

Keywords and Synonyms

Entropy coding; Statistical data compression

Problem Definition

Often it is desirable to encode a sequence of data efficiently to minimize the number of bits required to transmit or store the sequence. The sequence may be a file or message consisting of *symbols* (or *letters* or *characters*) taken from a fixed input alphabet, but more generally the sequence can be thought of as consisting of *events*, each taken from its own input set. Statistical data compression is concerned with encoding the data in a way that makes use of probability estimates of the events. Lossless compression has the property that the input sequence can be reconstructed exactly from the encoded sequence. Arithmetic coding is a nearly-optimal statistical coding technique that can produce a lossless encoding.

Problem (Statistical data compression)

INPUT: *A sequence of m events $a_1, a_2, \dots, a_m$. The ith event a_i is taken from a set of n distinct possible events $e_{i,1}, e_{i,2}, \dots, e_{i,n}$, with an accurate assessment of the probability distribution P_i of the events. The distributions P_i need not be the same for each event a_i.*

OUTPUT: *A succinct encoding of the events that can be decoded to recover exactly the original sequence of events.*

The goal is to achieve optimal or near-optimal encoding length. Shannon [10] proved that the smallest possible expected number of bits needed to encode the ith event is the *entropy* of P_i, denoted by

$$H(P_i) = \sum_{k=1}^{n} -p_{i,k} \log_2 p_{i,k}$$

where $p_{i,k}$ is the probability that e_k occurs as the ith event. An optimal code outputs $-\log_2 p$ bits to encode an event whose probability of occurrence is p.

The well-known Huffman codes [6] are optimal only among *prefix* (or *instantaneous*) codes, that is, those in which the encoding of one event can be decoded before encoding has begun for the next event. Hu–Tucker codes are prefix codes similar to Huffman codes, and are derived using a similar algorithm, with the added constraint that coded messages preserve the ordering of original messages.

When an instantaneous code is not needed, as is often the case, arithmetic coding provides a number of benefits, primarily by relaxing the constraint that the code lengths must be integers: 1) The code length is optimal ($-\log_2 p$ bits for an event with probability p), even when probabilities are not integer powers of $\frac{1}{2}$. 2) There is no loss of coding efficiency even for events with probability close to 1. 3) It is trivial to handle probability distributions that change from event to event. 4) The input message to output message ordering correspondence of Hu–Tucker coding can be obtained with minimal extra effort.

As an example, consider a 5-symbol input alphabet. Symbol probabilities, codes, and code lengths are given in Table 1.

The average code length is 2.13 bits per input symbol for the Huffman code, 2.22 bits per symbol for the Hu–

Arithmetic Coding for Data Compression, Table 1
Comparison of codes for Huffman coding, Hu-Tucker coding, and arithmetic coding for a sample 5-symbol alphabet

Symbol	Prob.		Huffman		Hu–Tucker		Arithmetic
e_k	p_k	$-\log_2 p_k$	Code	Length	Code	Length	Length
a	0.04	4.644	**1111**	4	**000**	3	4.644
b	0.18	2.474	**110**	3	**001**	3	2.474
c	0.43	1.218	**0**	1	**01**	2	1.218
d	0.15	2.737	**1110**	4	**10**	2	2.737
e	0.20	2.322	**10**	2	**11**	2	2.322

Tucker code, and 2.03 bits per symbol for arithmetic coding.

Key Results

In theory, arithmetic codes assign one "codeword" to each possible input sequence. The codewords consist of half-open subintervals of the half-open unit interval $[0, 1)$, and are expressed by specifying enough bits to distinguish the subinterval corresponding to the actual sequence from all other possible subintervals. Shorter codes correspond to larger subintervals and thus more probable input sequences. In practice, the subinterval is refined incrementally using the probabilities of the individual events, with bits being output as soon as they are known. Arithmetic codes almost always give better compression than prefix codes, but they lack the direct correspondence between the events in the input sequence and bits or groups of bits in the coded output file.

The algorithm for encoding a file using arithmetic coding works conceptually as follows:

1. The "current interval" $[L, H)$ is initialized to $[0, 1)$.
2. For each event in the file, two steps are performed.
   (a) Subdivide the current interval into subintervals, one for each possible event. The size of a event's subinterval is proportional to the estimated probability that the event will be the next event in the file, according to the model of the input.
   (b) Select the subinterval corresponding to the event that actually occurs next and make it the new current interval.
3. Output enough bits to distinguish the final current interval from all other possible final intervals.

The length of the final subinterval is clearly equal to the product of the probabilities of the individual events, which is the probability p of the particular overall sequence of events. It can be shown that $\lfloor -\log_2 p \rfloor + 2$ bits are enough to distinguish the file from all other possible files.

For finite-length files, it is necessary to indicate the end of the file. In arithmetic coding this can be done easily by introducing a special low-probability event that can be be injected into the input stream at any point. This adds only $O(\log m)$ bits to the encoded length of an m-symbol file.

In step 2, one needs to compute only the subinterval corresponding to the event a_i that actually occurs. To do this, it is convenient to use two "cumulative" probabilities: the cumulative probability $P_C = \sum_{k=1}^{i-1} p_k$ and the next-cumulative probability $P_N = P_C + p_i = \sum_{k=1}^{i} p_k$. The new subinterval is $[L + P_C(H - L), L + P_N(H - L))$. The need to maintain and supply cumulative probabilities requires the model to have a sophisticated data structure, such as that of Moffat [7], especially when many more than two events are possible.

Modeling

The goal of modeling for statistical data compression is to provide probability information to the coder. The modeling process consists of structural and probability estimation components; each may be adaptive (starting from a neutral model, gradually build up the structure and probabilities based on the events encountered), semi-adaptive (specify an initial model that describes the events to be encountered in the data, then modify the model during coding so that it describes only the events yet to be coded), or static (specify an initial model, and use it without modification during coding).

In addition there are two strategies for probability estimation. The first is to estimate each event's probability individually based on its frequency within the input sequence. The second is to estimate the probabilities collectively, assuming a probability distribution of a particular form and estimating the parameters of the distribution, either directly or indirectly. For direct estimation, the data can yield an estimate of the parameter (the variance, for instance). For indirect estimation [5], one can start with a small number of possible distributions and compute the code length that would be obtained with each; the one with the smallest code length is selected. This method is very general and can be used even for distributions from different families, without common parameters.

Arithmetic coding is often applied to text compression. The events are the symbols in the text file, and the model consists of the probabilities of the symbols considered in some context. The simplest model uses the overall frequencies of the symbols in the file as the probabilities; this is a zero-order Markov model, and its entropy is denoted H_0. The probabilities can be estimated adaptively starting with counts of 1 for all symbols and incrementing after each symbol is coded, or the symbol counts can be coded before coding the file itself and either modified during coding (a decrementing semi-adaptive code) or left unchanged (a static code). In all cases, the code length is independent of the order of the symbols in the file.

Theorem 1 *For all input files, the code length L_A of an adaptive code with initial 1-weights is the same as the code length L_{SD} of the semi-adaptive decrementing code plus the code length L_M of the input model encoded assuming that all symbol distributions are equally likely. This code length is less than $L_S = mH_0 + L_M$, the code length of a static code with the same input model. In other words, $L_A = L_{SD} + L_M < mH_0 + L_M = L_S$.*

It is possible to obtain considerably better text compression by using higher order Markov models. Cleary and Witten [2] were the first to do this with their PPM method. PPM requires adaptive modeling and coding of probabilities close to 1, and makes heavy use of arithmetic coding.

Implementation Issues

Incremental Output. The basic implementation of arithmetic coding described above has two major difficulties: the shrinking current interval requires the use of high precision arithmetic, and no output is produced until the entire file has been read. The most straightforward solution to both of these problems is to output each leading bit as soon as it is known, and then to double the length of the current interval so that it reflects only the unknown part of the final interval. Witten, Neal, and Cleary [11] add a clever mechanism for preventing the current interval from shrinking too much when the endpoints are close to $\frac{1}{2}$ but straddle $\frac{1}{2}$. In that case one does not yet know the next output bit, but whatever it is, the *following* bit will have the opposite value; one can merely keep track of that fact, and expand the current interval symmetrically about $\frac{1}{2}$. This follow-on procedure may be repeated any number of times, so the current interval size is always strictly longer than $\frac{1}{4}$.

Before [11] other mechanisms for incremental transmission and fixed precision arithmetic were developed through the years by a number of researchers beginning with Pasco [8]. The bit-stuffing idea of Langdon and others at IBM [9] that limits the propagation of carries in the additions serves a function similar to that of the follow-on procedure described above.

Use of Integer Arithmetic. In practice, the arithmetic can be done by storing the endpoints of the current interval as sufficiently large integers rather than in floating point or exact rational numbers. Instead of starting with the real interval $[0, 1)$, start with the integer interval $[0, N)$, N invariably being a power of 2. The subdivision process involves selecting non-overlapping integer intervals (of length at least 1) with lengths approximately proportional to the counts.

Limited-Precision Arithmetic Coding. Arithmetic coding as it is usually implemented is slow because of the multiplications (and in some implementations, divisions) required in subdividing the current interval according to the probability information. Since small errors in probability estimates cause very small increases in code length, introducing approximations into the arithmetic coding process in a controlled way can improve coding speed without significantly degrading compression performance. In the Q-Coder work at IBM [9], the time-consuming multiplications are replaced by additions and shifts, and low-order bits are ignored.

Howard and Vitter [4] describe a different approach to approximate arithmetic coding. The fractional bits characteristic of arithmetic coding are stored as state information in the coder. The idea, called *quasi-arithmetic coding*, is to reduce the number of possible states and replace arithmetic operations by table lookups; the lookup tables can be precomputed.

The number of possible states (after applying the interval expansion procedure) of an arithmetic coder using the integer interval $[0, N)$ is $3N^2/16$. The obvious way to reduce the number of states in order to make lookup tables practicable is to reduce N. Binary quasi-arithmetic coding causes an insignificant increase in the code length compared with pure arithmetic coding.

Theorem 2 *In a quasi-arithmetic coder based on full interval $[0, N)$, using correct probability estimates, and excluding very large and very small probabilities, the number of bits per input event by which the average code length obtained by the quasi-arithmetic coder exceeds that of an ex-*

act arithmetic coder is at most

$$\frac{4}{\ln 2}\left(\log_2 \frac{2}{e\ln 2}\right)\frac{1}{N}+O\left(\frac{1}{N^2}\right)\approx\frac{0.497}{N}+O\left(\frac{1}{N^2}\right),$$

and the fraction by which the average code length obtained by the quasi-arithmetic coder exceeds that of an exact arithmetic coder is at most

$$\left(\log_2 \frac{2}{e\ln 2}\right)\frac{1}{\log_2 N}+O\left(\frac{1}{(\log N)^2}\right)$$
$$\approx\frac{0.0861}{\log_2 N}+O\left(\frac{1}{(\log N)^2}\right).$$

General-purpose algorithms for parallel encoding and decoding using both Huffman and quasi-arithmetic coding are given in [3].

Applications

Arithmetic coding can be used in most applications of data compression. Its main usefulness is in obtaining maximum compression in conjunction with an adaptive model, or when the probability of one event is close to 1. Arithmetic coding has been used heavily in text compression. It has also been used in image compression in the JPEG international standards for image compression and is an essential part of the JBIG international standards for bilevel image compression. Many fast implementations of arithmetic coding, especially for a two-symbol alphabet, are covered by patents; considerable effort has been expended in adjusting the basic algorithm to avoid infringing those patents.

Open Problems

The technical problems with arithmetic coding itself have been completely solved. The remaining unresolved issues are concerned with modeling: decomposing an input data set into a sequence of events, the set of events possible at each point in the data set being described by a probability distribution suitable for input into the coder. The modeling issues are entirely application-specific.

Experimental Results

Some experimental results for the Calgary and Canterbury corpora are summarized in a report by Arnold and Bell [1].

Data Sets

Among the most widely used data sets suitable for research in arithmetic coding are: the Calgary Corpus: (ftp://ftp.cpsc.ucalgary.ca/pub/projects), the Canterbury Corpus (corpus.canterbury.ac.nz), and the Pizza&Chili Corpus (pizzachili.dcc.uchile.cl).

URL to Code

A number of implementations of arithmetic coding are available on the Compression Links Info page, www.compression-links.info/ArithmeticCoding. The code at the ucalgary.ca FTP site, based on [11], is especially useful for understanding arithmetic coding.

Cross References

► Boosting Textual Compression
► Burrows–Wheeler Transform

Recommended Reading

1. Arnold, R., Bell, T.: A corpus for the evaluation of lossless compression algorithms. In: Proceedings of the IEEE Data Compression Conference, Snowbird, Utah, March 1997, pp. 201–210
2. Cleary, J.G., Witten, I.H.: Data compression using adaptive coding and partial string matching. IEEE Transactions on Communications, COM–32, pp. 396–402 (1984)
3. Howard, P.G., Vitter, J.S.: Parallel lossless image compression using Huffman and arithmetic coding. In: Proceedings of the IEEE Data Compression Conference, Snowbird, Utah, March 1992, pp. 299–308
4. Howard, P.G., Vitter, J.S.: Practical implementations of arithmetic coding. In: Storer, J.A. (ed.) Images and Text Compression. Kluwer Academic Publishers, Norwell, Massachusetts (1992)
5. Howard, P.G., Vitter, J.S.: Fast and efficient lossless image compression. In: Proceedings of the IEEE Data Compression Conference, Snowbird, Utah, March 1993, pp. 351–360
6. Huffman, D.A.: A method for the construction of minimum redundancy codes. Proceedings of the Institute of Radio Engineers, 40, pp. 1098–1101 (1952)
7. Moffat, A.: An improved data structure for cumulative probability tables. Softw. Prac. Exp. **29**, 647–659 (1999)
8. Pasco, R.: Source Coding Algorithms for Fast Data Compression, Ph. D. thesis, Stanford University (1976)
9. Pennebaker, W.B., Mitchell, J.L., Langdon, G.G., Arps, R.B.: An overview of the basic principles of the Q-coder adaptive binary arithmetic coder. IBM J. Res. Develop. **32**, 717–726 (1988)
10. Shannon, C.E.: A mathematical theory of communication. Bell Syst. Tech. J. **27**, 398–403 (1948)
11. Witten, I.H., Neal, R.M., Cleary, J.G.: Arithmetic coding for data compression. Commun. ACM **30**, 520–540 (1987)

Assignment Problem

1955; Kuhn
1957; Munkres

SAMIR KHULLER
Department of Computer Science,
University of Maryland, College Park, MD, USA

Keywords and Synonyms

Weighted bipartite matching

Problem Definition

Assume that a complete bipartite graph, $G(X, Y, X \times Y)$, with weights $w(x, y)$ assigned to every edge (x, y) is given. A matching M is a subset of edges so that no two edges in M have a common vertex. A perfect matching is one in which all the nodes are matched. Assume that $|X| = |Y| = n$. The **weighted matching problem** is to find a matching with the greatest total weight, where $w(M) = \sum_{e \in M} w(e)$. Since G is a complete bipartite graph, it has a perfect matching. An algorithm that solves the weighted matching problem is due to Kuhn [4] and Munkres [6]. Assume that all edge weights are nonnegative.

Key Results

Define a *feasible vertex labeling* ℓ as a mapping from the set of vertices in G to the reals, where

$$\ell(x) + \ell(y) \geq w(x, y) .$$

Call $\ell(x)$ the label of vertex x. It is easy to compute a feasible vertex labeling as follows:

$$\forall y \in Y \quad \ell(y) = 0$$

and

$$\forall x \in X \quad \ell(x) = \max_{y \in Y} w(x, y) .$$

Define the **equality subgraph**, G_ℓ, to be the spanning subgraph of G, which includes all vertices of G but only those edges (x, y) that have weights such that

$$w(x, y) = \ell(x) + \ell(y) .$$

The connection between equality subgraphs and maximum-weighted matchings is provided by the following theorem.

Theorem 1 *If the equality subgraph, G_ℓ, has a perfect matching, M^*, then M^* is a maximum-weighted matching in G.*

In fact, note that the sum of the labels is an upper bound on the weight of the maximum-weighted perfect matching. The algorithm eventually finds a matching and a feasible labeling such that the weight of the matching is equal to the sum of all the labels.

High-Level Description

The above theorem is the basis of an algorithm for finding a maximum-weighted matching in a complete bipartite graph. Starting with a feasible labeling, compute the equality subgraph and then find a maximum matching in this subgraph (here one can ignore weights on edges). If the matching found is perfect, the process is done. If it is not perfect, more edges are added to the equality subgraph by revising the vertex labels. After adding edges to the equality subgraph, either the size of the matching goes up (an augmenting path is found) or the Hungarian tree continues to grow.[1] In the former case, the phase terminates and a new phase starts (since the matching size has gone up). In the latter case, the Hungarian tree, grows by adding new nodes to it, and clearly this cannot happen more than n times.

Let S be the set of free nodes in X. Grow Hungarian trees from each node in S. Let T be the nodes in Y encountered in the search for an augmenting path from nodes in S. Add all nodes from X that are encountered in the search to S.

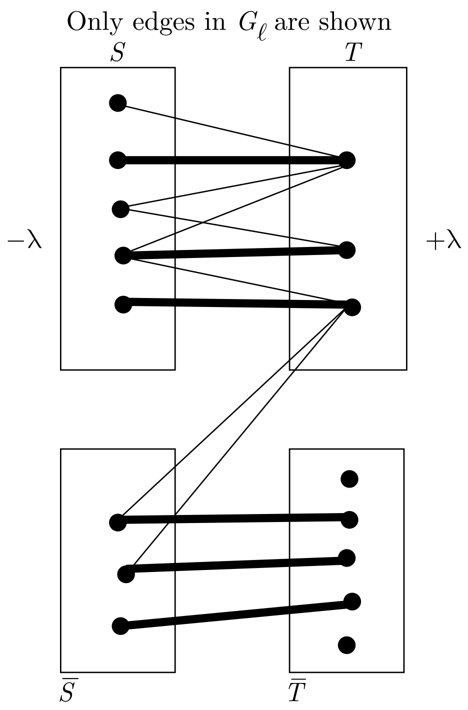
Note the following about this algorithm:

$$\overline{S} = X \setminus S .$$
$$\overline{T} = Y \setminus T .$$
$$|S| > |T| .$$

There are no edges from S to $\overline{T}$ since this would imply that one did not grow the Hungarian trees completely. As the Hungarian trees in are grown in G_ℓ, alternate nodes in the search are placed into S and T. To revise the labels, take the labels in S and start decreasing them uniformly (say, by λ), and at the same time increase the labels in T by λ. This ensures that the edges from S to T do not leave the equality subgraph (Fig. 1).

As the labels in S are decreased, edges (in G) from S to $\overline{T}$ will potentially enter the equality subgraph, G_ℓ. As we increase λ, at some point in time, an edge enters the equality subgraph. This is when one stops and updates the Hungarian tree. If the node from $\overline{T}$ added to T is matched to a node in $\overline{S}$, both these nodes are moved to S and T, which yields a larger Hungarian tree. If the node from $\overline{T}$ is free, an augmenting path is found and the phase is complete. One phase consists of those steps taken between increases in the size of the matching. There are at most n phases, where n is the number of vertices in G (since in each phase

[1] This is the structure of explored edges when one starts BFS simultaneously from all free nodes in S. When one reaches a matched node in T, one only explores the matched edge; however, all edges incident to nodes in S are explored.



Assignment Problem, Figure 1
Sets *S* and *T* as maintained by the algorithm

the size of the matching increases by 1). Within each phase the size of the Hungarian tree is increased at most n times. It is clear that in $O(n^2)$ time one can figure out which edge from S to $\overline{T}$ is the first to enter the equality subgraph (one simply scans all the edges). This yields an $O(n^4)$ bound on the total running time. How to implement it in $O(n^3)$ time is now shown.

More Efficient Implementation

Define the slack of an edge as follows:

$$slack(x, y) = \ell(x) + \ell(y) - w(x, y) .$$

Then

$$\lambda = \min_{x \in S, y \in \overline{T}} slack(x, y) .$$

Naively, the calculation of λ requires $O(n^2)$ time. For every vertex $y \in \overline{T}$, keep track of the edge with the smallest slack, i. e.,

$$slack[y] = \min_{x \in S} slack(x, y) .$$

The computation of $slack[y]$ (for all $y \in \overline{T}$) requires $O(n^2)$ time at the start of a phase. As the phase progresses, it is easy to update all the *slack* values in $O(n)$ time since all of them change by the same amount (the labels of the vertices in S are going down uniformly). Whenever a node u is moved from $\overline{S}$ to S one must recompute the slacks of the nodes in $\overline{T}$, requiring $O(n)$ time. But a node can be moved from $\overline{S}$ to S at most n times.

Thus each phase can be implemented in $O(n^2)$ time. Since there are n phases, this gives a running time of $O(n^3)$. For sparse graphs, there is a way to implement the algorithm in $O(n(m + n \log n))$ time using min cost flows [1], where m is the number of edges.

Applications

There are numerous applications of biparitite matching, for example, scheduling unit-length jobs with integer release times and deadlines, even with time-dependent penalties.

Open Problems

Obtaining a linear, or close to linear, time algorithm.

Recommended Reading

Several books on combinatorial optimization describe algorithms for weighted bipartite matching (see [2,5]). See also Gabow's paper [3].

1. Ahuja, R., Magnanti, T., Orlin, J.: Network Flows: Theory, Algorithms and Applications. Prentice Hall, Englewood Cliffs (1993)
2. Cook, W., Cunningham, W., Pulleyblank, W., Schrijver, A.: Combinatorial Optimization. Wiley, New York (1998)
3. Gabow, H.: Data structures for weighted matching and nearest common ancestors with linking. In: Symp. on Discrete Algorithms, 1990, pp. 434–443
4. Kuhn, H.: The Hungarian method for the assignment problem. Naval Res. Logist. Quart. **2**, 83–97 (1955)
5. Lawler, E.: Combinatorial Optimization: Networks and Matroids. Holt, Rinehart and Winston (1976)
6. Munkres, J.: Algorithms for the assignment and transportation problems. J. Soc. Ind. Appl. Math. **5**, 32–38 (1957)

Asynchronous Consensus Impossibility

1985; Fischer, Lynch, Paterson

Maurice Herlihy
Department of Computer Science, Brown University, Providence, RI, USA

Keywords and Synonyms

Wait-free consensus; Agreement

Problem Definition

Consider a distributed system consisting of a set of *processes* that communicate by sending and receiving messages. The network is a multiset of messages, where each message is addressed to some process. A process is a state machine that can take three kinds of *steps*.

- In a *send* step, a process places a message in the network.
- In a *receive* step, a process A either reads and removes from the network a message addressed to A, or it reads a distinguished *null* value, leaving the network unchanged. If a message addressed to A is placed in the network, and if A subsequently performs an infinite number of receive steps, then A will eventually receive that message.
- In a *computation* state, a process changes state without communicating with any other process.

Processes are *asynchronous*: there is no bound on their relative speeds. Processes can *crash*: they can simply halt and take no more steps. This article considers executions in which at most one process crashes.

In the *consensus* problem, each process starts with a private *input* value, communicates with the others, and then halts with a *decision* value. These values must satisfy the following properties:

- *Agreement:* all processes' decision values must agree.
- *Validity:* every decision value must be some process' input.
- *Termination:* every non-fault process must decide in a finite number of steps.

Fischer, Lynch, and Paterson showed that there is no protocol that solves consensus in any asynchronous message-passing system where even a single process can fail. This result is one of the most influential results in Distributed Computing, laying the foundations for a number of subsequent research efforts.

Terminology

Without loss of generality, one can restrict attention to *binary* consensus, where the inputs are 0 or 1. A *protocol state* consists of the states of the processes and the multiset of messages in transit in the network. An *initial state* is a protocol state before any process has moved, and a *final state* is a protocol state after all processes have finished. The *decision value* of any final state is the value decided by all processes in that state.

Any terminating protocol's set of possible states forms a tree, where each node represents a possible protocol state, and each edge represents a possible step by some process. Because the protocol must terminate, the tree is finite. Each leaf node represents a final protocol state with decision value either 0 or 1.

A *bivalent* protocol state is one in which the eventual decision value is not yet fixed. From any bivalent state, there is an execution in which the eventual decision value is 0, and another in which it is 1. A *univalent* protocol state is one in which the outcome is fixed. Every execution starting from a univalent state decides the same value. A *1-valent* protocol state is univalent with eventual decision value 1, and similarly for a *0-valent* state.

A protocol state is *critical* if

- It is bivalent, and
- If any process takes a step, the protocol state becomes univalent.

Key Results

Lemma 1 *Every consensus protocol has a bivalent initial state.*

Proof Assume, by way of contradiction, that there exists a consensus protocol for $(n+1)$ threads $A_0, \cdots, A_n$ in which every initial state is univalent. Let s_i be the initial state where processes $A_i, \cdots, A_n$ have input 0 and $A_0, \ldots, A_{i-1}$ have input 1. Clearly, s_0 is 0-valent: all processes have input 0, so all must decide 0 by the validity condition. If s_i is 0-valent, so is s_{i+1}. These states differ only in the input to process A_i : 0 in s_i, and 1 in s_{i+1}. Any execution starting from s_i in which A_i halts before taking any steps is indistinguishable from an execution starting from s_{i+1} in which A_i halts before taking any steps. Since processes must decide 0 in the first execution, they must decide 1 in the second. Since there is one execution starting from s_{i+1} that decides 0, and since s_{i+1} is univalent by hypothesis, s_{i+1} is 0-valent. It follows that the state s_{n+1}, in which all processes start with input 1, is 0-valent, a contradiction. □

Lemma 2 *Every consensus protocol has a critical state.*

Proof by contradiction. By Lemma 1, the protocol has a bivalent initial state. Start the protocol in this state. Repeatedly choose a process whose next step leaves the protocol in a bivalent state, and let that process take a step. Either the protocol runs forever, violating the termination condition, or the protocol eventually enters a critical state. □

Theorem 3 *There is no consensus protocol for an asynchronous message-passing system where a single process can crash.*

Proof Assume by way of contradiction that such a protocol exists. Run the protocol until it reaches a critical state

s. There must be two processes A and B such that A's next step carries the protocol to a 0-valent state, and B's next step carries the protocol to a 1-valent state.

Starting from s, let s_A be the state reached if A takes the first step, s_B if B takes the first step, s_{AB} if A takes a step followed by B, and so on. States s_A and s_{AB} are 0-valent, while s_B and s_{BA} are 1-valent. The rest is a case analysis.

Of all the possible pairs of steps A and B could be about to execute, most of them *commute*: states s_{AB} and s_{BA} are identical, which is a contradiction because they have different valences.

The only pair of steps that do not commute occurs when A is about to send a message to B (or vice versa). Let s_{AB} be the state resulting if A sends a message to B and B then receives it, and let s_{BA} be the state resulting if B receives a different message (or *null*) and then A sends its message to B. Note that every process other than B has the same local state in s_{AB} and s_{BA}. Consider an execution starting from s_{AB} in which every process other than B takes steps in round-robin order. Because s_{AB} is 0-valent, they will eventually decide 0. Next, consider an execution starting from s_{BA} in which every process other than B takes steps in round-robin order. Because s_{BA} is 1-valent, they will eventually decide 1. But all processes other than B have the same local states at the end of each execution, so they cannot decide different values, a contradiction. □

In the proof of this theorem, and in the proofs of the preceding lemmas, we construct scenarios where at most a single process is delayed. As a result, this impossibility result holds for any system where a single process can fail undetectably.

Applications

The consensus problem is a key tool for understanding the power of various asynchronous models of computation.

Open Problems

There are many open problems concerning the solvability of consensus in other models, or with restrictions on inputs.

Related Work

The original paper by Fischer, Lynch, and Paterson [8] is still a model of clarity.

Many researchers have examined alternative models of computation in which consensus can be solved. Dolev, Dwork, and Stockmeyer [5] examine a variety of alternative message-passing models, identifying the precise assumptions needed to make consensus possible. Dwork, Lynch, and Stockmeyer [6] derive upper and lower bounds for a semi-synchronous model where there is an upper and lower bound on message delivery time. Ben-Or [1] showed that introducing randomization makes consensus possible in an asynchronous message-passing system. Chandra and Toueg [3] showed that consensus becomes possible if in the presence of an oracle that can (unreliably) detect when a process has crashed. Each of the papers cited here has inspired many follow-up papers. A good place to start is the excellent survey by Fich and Ruppert [7].

A protocol is *wait-free* if it tolerates failures by all but one of the participants. A concurrent object implementation is *linearizable* if each method call seems to take effect instantaneously at some point between the method's invocation and response. Herlihy [9] showed that shared-memory objects can each be assigned a *consensus number*, which is the maximum number of processes for which there exists a wait-free consensus protocol using a combination of read-write memory and the objects in question. Consensus numbers induce an infinite hierarchy on objects, where (simplifying somewhat) higher objects are more powerful than lower objects. In a system of n or more concurrent processes, it is impossible to construct a lock-free implementation of an object with consensus number n from an object with a lower consensus number. On the other hand, any object with consensus number n is *universal* in a system of n or fewer processes: it can be used to construct a wait-free linearizable implementation of any object.

In 1990, Chaudhuri [4] introduced the *k-set agreement* problem (sometimes called *k-set consensus*, which generalizes consensus by allowing k or fewer distinct decision values to be chosen. In particular, 1-set agreement is consensus. The question whether k-set agreement can be solved in asynchronous message-passing models was open for several years, until three independent groups [2,10,11] showed that no protocol exists.

Cross References

► Linearizability
► Topology Approach in Distributed Computing

Recommended Reading

1. Ben-Or, M.: Another advantage of free choice (extended abstract): Completely asynchronous agreement protocols. In: PODC '83: Proceedings of the second annual ACM symposium on Principles of distributed computing, pp. 27–30. ACM Press, New York (1983)

2. Borowsky, E., Gafni, E.: Generalized FLP impossibility result for *t*-resilient asynchronous computations. In: Proceedings of the 1993 ACM Symposium on Theory of Computing, May 1993. pp. 206–215
3. Chandra, T.D., Toueg, S.: Unreliable failure detectors for reliable distributed systems. J. ACM **43**(2), 225–267 (1996)
4. Chaudhuri, S.: Agreement is harder than consensus: Set consensus problems in totally asynchronous systems. In: Proceedings Of The Ninth Annual ACM Symposium On Principles of Distributed Computing, August 1990. pp. 311–234
5. Chandhuri, S.: More Choices Allow More Faults: Set Consensus Problems in Totally Asynchronous Systems. Inf. Comput. **105**(1), 132–158, July 1993
6. Dwork, C., Lynch, N., Stockmeyer, L.: Consensus in the presence of partial synchrony. J. ACM **35**(2), 288–323 (1988)
7. Fich, F., Ruppert, E.: Hundreds of impossibility results for distributed computing. Distrib. Comput. **16**(2–3), 121–163 (2003)
8. Fischer, M., Lynch, N., Paterson, M.: Impossibility of distributed consensus with one faulty process. J. ACM **32**(2), 374–382 (1985)
9. Herlihy, M.: Wait-free synchronization. ACM Trans. Program. Lang. Syst. (TOPLAS) **13**(1), 124–149 (1991)
10. Herlihy, M., Shavit, N.: The topological structure of asynchronous computability. J. ACM **46**(6), 858–923 (1999)
11. Saks, M.E., Zaharoglou, F.: Wait-free k-set agreement is impossible: The topology of public knowledge. SIAM J. Comput. **29**(5), 1449–1483 (2000)

Atomic Broadcast

1995; Cristian, Aghili, Strong, Dolev

XAVIER DÉFAGO
School of Information Science, Japan Advanced Institute of Science and Technology (JAIST),
Ishikawa, Japan

Keywords and Synonyms

Atomic multicast; Total order broadcast; Total order multicast

Problem Definition

The problem is concerned with allowing a set of processes to concurrently broadcast messages while ensuring that all destinations consistently deliver them in the exact same sequence, in spite of the possible presence of a number of faulty processes.

The work of Cristian, Aghili, Strong, and Dolev [7] considers the problem of atomic broadcast in a system with approximately synchronized clocks and bounded transmission and processing delays. They present successive extensions of an algorithm to tolerate a bounded number of omission, timing, or Byzantine failures, respectively.

Related Work

The work presented in this entry originally appeared as a widely distributed conference contribution [6], over a decade before being published in a journal [7], at which time the work was well-known in the research community. Since there was no significant change in the algorithms, the historical context considered here is hence with respect to the earlier version.

Lamport [11] proposed one of the first published algorithms to solve the problem of ordering broadcast messages in a distributed systems. That algorithm, presented as the core of a mutual exclusion algorithm, operates in a fully asynchronous system (i. e., a system in which there are no bounds on processor speed or communication delays), but does not tolerate failures. Although the algorithms presented here rely on physical clocks rather than Lamport's logical clocks, the principle used for ordering messages is essentially the same: message carry a timestamp of their sending time; messages are delivered in increasing order of the timestamp, using the sending processor name for messages with equal timestamps.

At roughly the same period as the initial publication of the work of Cristian et al. [6], Chang and Maxemchuck [3] proposed an atomic broadcast protocol based on a token passing protocol, and tolerant to crash failures of processors. Also, Carr [1] proposed the Tandem global update protocol, tolerant to crash failures of processors.

Cristian [5] later proposed an extension to the omission-tolerant algorithm presented here, under the assumption that the communication system consists of $f + 1$ independent broadcast channels (where f is the maximal number of faulty processors). Compared with the more general protocol presented here, its extension generates considerably fewer messages.

Since the work of Cristian, Aghili, Strong, and Dolev [7], much has been published on the problem of atomic broadcast (and its numerous variants). For further reading, Défago, Schiper, and Urbán [8] surveyed more than sixty different algorithms to solve the problem, classifying them into five different classes and twelve variants. That survey also reviews many alternative definitions and references about two hundred articles related to this subject. This is still a very active research area, with many new results being published each year.

Hadzilacos and Toueg [10] provide a systematic classification of specifications for variants of atomic broadcast as well as other broadcast problems, such as reliable broadcast, FIFO broadcast, or causal broadcast.

Chandra and Toueg [2] proved the equivalence between atomic broadcast and the *consensus* problem. Thus, any application solved by a consensus can also be solved by atomic broadcast and vice-versa. Similarly, impossibility results apply equally to both problems. For instance, it is well-known that consensus, thus atomic broadcast, cannot be solved deterministically in an asynchronous system with the presence of a faulty process [9].

Notations and Assumptions

The system G consists of n distributed processors and m point-to-point communication links. A link does not necessarily exists between every pair of processors, but it is assumed that the communication network remains connected even in the face of faults (whether processors or links). All processors have distinct names and there exists a total order on them (e. g., lexicographic order).

A component (link or processor) is said to be *correct* if its behavior is consistent with its specification, and *faulty* otherwise. The paper considers three classes of component failures, namely, omission, timing, and Byzantine failures.

- An *omission* failure occurs when the faulty component fails to provide the specified output (e. g., loss of a message).
- A *timing* failure occurs when the faulty component omits a specified output, or provides it either too early or too late.
- A *Byzantine* failure [12] occurs when the component does not behave according to its specification, for instance, by providing output different from the one specified. In particular, the paper considers authentication-detectable Byzantine failures, that is, ones that are detectable using a message authentication protocol, such as error correction codes or digital signatures.

Each processor p has access to a local clock C_p with the properties that (1) two separate clock readings yield different values, and (2) clocks are ε-synchronized, meaning that, at any real time t, the deviation in readings of the clocks of any two processors p and q is at most ε.

In addition, transmission and processing delays, as measured on the clock of a correct processor, are bounded by a known constant δ. This bound accounts not only for delays in transmission and processing, but also for delays due to scheduling, overload, clock drift or adjustments. This is called a synchronous system model.

The diffusion time $d\delta$ is the time necessary to propagate information to all correct processes, in a surviving network of diameter d with the presence of a most π processor failures and λ link failures.

Problem Definition

The problem of atomic broadcast is defined in a synchronous system model as a broadcast primitive which satisfies the following three properties: atomicity, order, and termination.

Problem 1 (Atomic broadcast)
Input*: A stream of messages broadcast by n concurrent processors, some of which may be faulty.*
Output*: The messages delivered in sequence, with the following properties:*

1. *Atomicity: if any correct processor delivers an update at time U on its clock, then that update was initiated by some processor and is delivered by each correct processor at time U on its clock.*
2. *Order: all updates delivered by correct processors are delivered in the same order by each correct processor.*
3. *Termination: every update whose broadcast is initiated by a correct processor at time T on its clock is delivered at all correct processors at time $T + \Delta$ on their clock.*

Nowadays, problem definitions for atomic broadcast that do not explicitly refer to physical time are often preferred. Many variants of time-free definitions are reviewed by Hadzilacos and Toueg [10] and Défago et al. [8]. One such alternate definition is presented below, with the terminology adapted to the context of this entry.

Problem 2 (Total order broadcast)
Input*: A stream of messages broadcast by n concurrent processors, some of which may be faulty.*
Output*: The messages delivered in sequence, with the following properties:*

1. *Validity: if a correct processor broadcasts a message m, then it eventually delivers m.*
2. *Uniform agreement: if a processor delivers a message m, then all correct processors eventually deliver m.*
3. *Uniform integrity: for any message m, every processor delivers m at most once, and only if m was previously broadcast by its sending processor.*
4. *Gap-free uniform total order: if some processor delivers message m' after message m, then a processor delivers m' only after it has delivered m.*

Key Results

The paper presents three algorithms for solving the problem of atomic broadcast, each under an increasingly demanding failure model, namely, omission, timing, and

Byzantine failures. Each protocol is actually an extension of the previous one.

All three protocols are based on a classical flooding, or information diffusion, algorithm [14]. Every message carries its initiation timestamp T, the name of the initiating processor s, and an update σ. A message is then uniquely identified by (s, T). Then, the basic protocol is simple. Each processor logs every message it receives until it is delivered. When it receives a message that was never seen before, it forwards that message to all other neighbor processors.

Atomic Broadcast for Omission Failures

The first atomic broadcast protocol, supporting omission failures, considers a termination time Δ_o as follows.

$$\Delta_o = \pi\delta + d\delta + \varepsilon \,. \tag{1}$$

The delivery deadline $T + \Delta_o$ is the time by which a processor can be sure that it has received copies of every message with timestamp T (or earlier) that could have been received by some correct process.

The protocol then works as follows. When a processor initiates an atomic broadcast, it propagates that message, similar to the diffusion algorithm described above. The main exception is that every message received after the local clock exceeds the delivery deadline of that message, is discarded. Then, at local time $T + \Delta_o$, a processor delivers all messages timestamped with T, in order of the name of the sending processor. Finally, it discards all copies of the messages from its logs.

Atomic Broadcast for Timing Failures

The second protocol extends the first one by introducing a hop count (i. e., a counter incremented each time a message is relayed) to the messages. With this information, each relaying processor can determine when a message is timely, that is, if a message timestamped T with hop count h is received at time U then the following condition must hold.

$$T - h\varepsilon < U < T + h(\delta + \varepsilon) \,. \tag{2}$$

Before relaying a message, each processor checks the acceptance test above and discard the message if it does not satisfy it. The termination time Δ_t of the protocol for timing failures is as follows.

$$\Delta_t = \pi(\delta + \varepsilon) + d\delta + \varepsilon \,. \tag{3}$$

The authors point out that discarding early messages is not necessary for correctness, but ensures that correct processors keep messages in their log for a bounded amount of time.

Atomic Broadcast for Byzantine Failures

Given some text, every processor is assumed to be able to generate a signature for it, that cannot be faked by other processors. Furthermore, every processor knows the name of every other processors in the network, and has the ability to verify the authenticity of their signature.

Under the above assumptions, the third protocol extends the second one by adding signatures to the messages. To prevent a Byzantine processor (or link) from tampering with the hop count, a message is co-signed by every processor that relays it. For instance, a message signed by k processors $p_1, \ldots, p_k$ is as follows.

$$\left(\mathit{relayed}, \ldots \left(\mathit{relayed}, \left(\mathit{first}, T, \sigma, p_1, s_1\right), p_2, s_2\right), \ldots p_k, s_k\right)$$

Where σ is the update, T the timestamp, p_1 the message source, and s_i the signature generated by processor p_i. Any message for which one of the signature cannot be authenticated is simply discarded. Also, if several updates initiated by the same processor p carry the same timestamp, this indicates that p is faulty and the corresponding updates are discarded. The remainder of the protocol is the same as the second one, where the number of hops is given by the number of signatures. The termination time Δ_b is also as follows.

$$\Delta_b = \pi(\delta + \varepsilon) + d\delta + \varepsilon \,. \tag{4}$$

The authors insist however that, in this case, the transmission time δ must be considerably larger than in the previous case, since it must account for the time spent in generating and verifying the digital signatures; usually a costly operation.

Bounds

In addition to the three protocols presented above and their correctness, Cristian *et al.* [7] prove the following two lower bounds on the termination time of atomic broadcast protocols.

Theorem 1 *If the communication network G requires x steps, then any atomic broadcast protocol tolerant of up to π processor and λ link omission failures has a termination time of at least $x\delta + \varepsilon$.*

Theorem 2 *Any atomic broadcast protocol for a Hamiltonian network with n processors that tolerate $n - 2$ authentication-detectable Byzantine processor failures cannot have a termination time smaller than $(n - 1)(\delta + \varepsilon)$.*

Applications

The main motivation for considering this problem is its use as the cornerstone for ensuring fault-tolerance through process replication. In particular, the authors consider a *synchronous replicated storage*, which they define as a distributed and resilient storage system that displays the same content at every correct physical processor at any clock time. Using atomic broadcast to deliver updates ensures that all updates are applied at all correct processors in the same order. Thus, provided that the replicas are initially consistent, they will remain consistent. This technique, called *state-machine replication* [11,13] or also *active replication*, is widely used in practice as a means of supporting fault-tolerance in distributed systems.

In contrast, Cristian et al. [7] consider atomic broadcast in a *synchronous* system with bounded transmission and processing delays. Their work was motivated by the implementation of a highly-available replicated storage system, with tightly coupled processors running a real-time operating system.

Atomic broadcast has been used as a support for the replication of running processes in real-time systems or, with the problem reformulated to isolate explicit timing requirements, has also been used as a support for fault-tolerance and replication in many group communication toolkits (see survey of Chockler et al. [4]).

In addition, atomic broadcast has been used for the replication of database systems, as a means to reduce the synchronization between the replicas. Wiesmann and Schiper [15] have compared different database replication and transaction processing approaches based on atomic broadcast, showing interesting performance gains.

Cross References

▶ Asynchronous Consensus Impossibility
▶ Causal Order, Logical Clocks, State Machine Replication
▶ Clock Synchronization
▶ Failure Detectors

Recommended Reading

1. Carr, R.: The Tandem global update protocol. Tandem Syst. Rev. **1**, 74–85 (1985)
2. Chandra, T.D., Toueg, S.: Unreliable failure detectors for reliable distributed systems. J. ACM **43**, 225–267 (1996)
3. Chang, J.-M., Maxemchuk, N.F.: Reliable broadcast protocols. ACM Trans. Comput. Syst. **2**, 251–273 (1984)
4. Chockler, G., Keidar, I., Vitenberg, R.: Group communication specifications: A comprehensive study. ACM Comput. Surv. **33**, 427–469 (2001)
5. Cristian, F.: Synchronous atomic broadcast for redundant broadcast channels. Real-Time Syst. **2**, 195–212 (1990)
6. Cristian, F., Aghili, H., Strong, R., Dolev, D.: Atomic Broadcast: From simple message diffusion to Byzantine agreement. In: Proc. 15th Intl. Symp. on Fault-Tolerant Computing (FTCS-15), Ann Arbor, June 1985 pp. 200–206. IEEE Computer Society Press
7. Cristian, F., Aghili, H., Strong, R., Dolev, D.: Atomic broadcast: From simple message diffusion to Byzantine agreement. Inform. Comput. **118**, 158–179 (1995)
8. Défago, X., Schiper, A., Urbán, P.: Total order broadcast and multicast algorithms: Taxonomy and survey. ACM Comput. Surveys **36**, 372–421 (2004)
9. Fischer, M.J., Lynch, N.A., Paterson, M.S.: Impossibility of distributed consensus with one faulty process. J. ACM **32**, 374–382 (1985)
10. Hadzilacos, V., Toueg, S.: Fault-tolerant broadcasts and related problems. In: Mullender, S. (ed.) Distributed Systems, 2nd edn., pp. 97–146. ACM Press Books, Addison-Wesley (1993). Extended version appeared as Cornell Univ. TR 94-1425
11. Lamport, L.: Time, clocks, and the ordering of events in a distributed system. Comm. ACM **21**, 558–565 (1978)
12. Lamport, L., Shostak, R., Pease, M.: The Byzantine generals problem. ACM Trans. Prog. Lang. Syst. **4**, 382–401 (1982)
13. Schneider, F.B.: Implementing fault-tolerant services using the state machine approach: a tutorial. ACM Comput. Surveys **22**, 299–319 (1990)
14. Segall, A.: Distributed network protocols. IEEE Trans. Inform. Theory **29**, 23–35 (1983)
15. Wiesmann, M., Schiper, A.: Comparison of database replication techniques based on total order broadcast. IEEE Trans. Knowl. Data Eng. **17**, 551–566 (2005)

Atomicity

▶ Best Response Algorithms for Selfish Routing
▶ Linearizability
▶ Selfish Unsplittable Flows: Algorithms for Pure Equilibria
▶ Snapshots in Shared Memory

Atomic Multicast

▶ Atomic Broadcast

Atomic Network Congestion Games

▶ Selfish Unsplittable Flows: Algorithms for Pure Equilibria

Atomic Scan

▶ Snapshots in Shared Memory

Atomic Selfish Flows

▶ Best Response Algorithms for Selfish Routing

Attribute-Efficient Learning

1987; Littlestone

JYRKI KIVINEN
Department of Computer Science, University of Helsinki, Helsinki, Finland

Keywords and Synonyms

Learning with irrelevant attributes

Problem Definition

Given here is a basic formulation using the *online mistake bound* model, which was used by Littlestone [9] in his seminal work.

Fix a class C of Boolean functions over n variables. To start a learning scenario, a *target function* $f_* \in C$ is chosen but not revealed to the *learning algorithm*. Learning then proceeds in a sequence of *trials*. At trial t, an input $\boldsymbol{x}_t \in \{0,1\}^n$ is first given to the learning algorithm. The learning algorithm then produces its *prediction* $\hat{y}_t$, which is its guess as to the unknown value $f_*(x_t)$. The correct value $y_t = f_*(x_t)$ is then revealed to the learner. If $y_t \neq \hat{y}_t$, the learning algorithm made a *mistake*. The learning algorithm learns C with mistake bound m, if the number of mistakes never exceeds m, no matter how many trials are made and how f_* and $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots$ are chosen.

Variable (or attribute) X_i is *relevant* for function $f\colon \{0,1\}^n \to \{0,1\}$ if $f(x_1,\ldots,x_i,\ldots,x_n) \neq f(x_1,\ldots,1-x_i,\ldots,x_n)$ holds for some $\vec{x} \in 0,1^n$. Suppose now that for some $k \leq$ n, every function $f \in C$ has at most k relevant variables. It is said that a learning algorithm learns class C *attribute-efficiently*, if it learns C with a mistake bound polynomial in k and $\log n$. Additionally, the computation time for each trial is usually required to be polynomial in n.

Key Results

The main part of current research of attribute-efficient learning stems from Littlestones Winnow algorithm [9].

The basic version of Winnow maintains a weight vector $\boldsymbol{w}_t = (w_{t,1},\ldots,w_{t,n}) \in \mathbb{R}^n$. The prediction for input $\boldsymbol{x}_t \in \{0,1\}^n$ is given by

$$\hat{y}_t = \operatorname{sign}\left(\sum_{i=1}^{n} w_{t,i}x_{t,i} - \theta\right)$$

where θ is a parameter of the algorithm. Initially $\boldsymbol{w}_1 = (1,\ldots,1)$, and after trial t each component $w_{t,i}$ is updated according to

$$w_{t+1,i} = \begin{cases} \alpha w_{t,i} & \text{if } y_t = 1,\ \hat{y}_t = 0 \text{ and } x_{t,i} = 1 \\ w_{t,i}/\alpha & \text{if } y_t = 0,\ \hat{y}_t = 1 \text{ and } x_{t,i} = 1 \\ w_{t,i} & \text{otherwise} \end{cases} \quad (1)$$

where $\alpha > 1$ is a learning rate parameter.

Littlestone's basic result is that with a suitable choice of θ and α, Winnow learns the class of monotone k-literal disjunctions with mistake bound $O(k \log n)$. Since the algorithm changes its weights only when a mistake occurs, this bound also guarantees that the weights remain small enough for computation times to remain polynomial in n. With simple transformations, Winnow also yields attribute-efficient learning algorithms for general disjunctions and conjunctions. Various subclasses of DNF formulas and decision lists [8] can be learned, too.

Winnow is quite robust against noise, i. e., errors in input data. This is extremely important for practical applications. Remove now the assumption about a target function $f_* \in C$ satisfying $y_t = f_*(x_t)$ for all t. Define *attribute error* of a pair $(\boldsymbol{x}, y)$ with respect to a function f as the minimum Hamming distance between $\boldsymbol{x}$ and $\boldsymbol{x}'$ such that $f(\boldsymbol{x}') = y$. The attribute error of a sequence of trials with respect to f is the sum of attribute errors of the individual pairs $(\boldsymbol{x}_t, y_t)$. Assuming the sequence of trials has attribute error at most A with respect to some k-literal disjunction, Auer and Warmuth [1] show that Winnow makes $O(A + k \log n)$ mistakes. The noisy scenario can also be analyzed in terms of *hinge loss* [5].

The update rule (1) has served as a model for a whole family of *multiplicative update algorithms*. For example, Kivinen and Warmuth [7] introduce the Exponentiated Gradient algorithm, which is essentially Winnow modified for continuous-valued prediction, and show how it can be motivated by a relative entropy minimization principle.

Consider a function class C where each function can be encoded using $O(p(k) \log n)$ bits for some polynomial p. An example would be Boolean formulas with k relevant variables, when the size of the formula is restricted to $p(k)$ ignoring the size taken by the variables. The cardinality of C is then $|C| = 2^{O(p(k) \log n)}$. The classical Halving

Algorithm (see [9] for discussion and references) learns any class consisting of m Boolean functions with mistake bound $\log_2 m$, and would thus provide an attribute-efficient algorithm for such a class C. However, the running time would not be polynomial. Another serious drawback would be that the Halving Algorithm does not tolerate any noise. Interestingly, a multiplicative update similar to (1) has been used in Littlestone and Warmuth's Weighted Majority Algorithm [10], and also Vovk's Aggregating Algorithm [14], to produce a noise-tolerant generalization of the Halving Algorithm.

Attribute-efficient learning has also been studied in other learning models than the mistake bound model, such as Probably Approximately Correct learning [4], learning with uniform distribution [12], and learning with membership queries [3]. The idea has been further developed into learning with a potentially infinite number of attributes [2].

Applications

Attribute-efficient algorithms for simple function classes have a potentially interesting application as a component in learning more complex function classes. For example, any monotone k-term DNF formula over variables $x_1,\ldots,x_n$ can be represented as a monotone k-literal disjunction over 2^n variables z_A, where $z_A = \prod_{i\in A} x_i$ for $A \subseteq \{1,\ldots,n\}$ is defined. Running Winnow with the transformed inputs $z \in \{0,1\}^{2^n}$ would give a mistake bound $O(k \log 2^n) = O(kn)$. Unfortunately the running time would be linear in 2^n, at least for a naive implementation. Khardon et al. [6] provide discouraging computational hardness results for this potential application.

Online learning algorithms have a natural application domain in signal processing. In this setting, the sender emits a true signal y_t at time t, for $t = 1, 2, 3, \ldots$. At some later time $(t+d)$, a receiver receives a signal z_t, which is a sum of the original signal y_t and various echoes of earlier signals $y_{t'}$, $t' < t$, all distorted by random noise. The task is to recover the true signal y_t based on received signals $z_t, z_{t-1}, \ldots, z_{t-l}$ over some time window l. Currently attribute-efficient algorithms are not used for such tasks, but see [11] for preliminary results.

Attribute-efficient learning algorithms are similar in spirit to statistical methods that find sparse models. In particular, statistical algorithms that use L_1 regularization are closely related to multiplicative algorithms such as Winnow and Exponentiated Gradient. In contrast, more classical L_2 regularization leads to algorithms that are not attribute-efficient [13].

Cross References

▶ Boosting Textual Compression
▶ Learning DNF Formulas

Recommended Reading

1. Auer, P., Warmuth, M.K.: Tracking the best disjunction. Mach. Learn. **32**(2), 127–150 (1998)
2. Blum, A., Hellerstein, L., Littlestone, N.: Learning in the presence of finitely or infinitely many irrelevant attributes. J. Comp. Syst. Sci. **50**(1), 32–40 (1995)
3. Bshouty, N., Hellerstein, L.: Attribute-efficient learning in query and mistake-bound models. J. Comp. Syst. Sci. **56**(3), 310–319 (1998)
4. Dhagat, A., Hellerstein, L.: PAC learning with irrelevant attributes. In: Proceedings of the 35th Annual Symposium on Foundations of Computer Science, Santa Fe, pp 64–74. IEEE Computer Society, Los Alamitos (1994)
5. Gentile, C., Warmuth, M.K.: Linear hinge loss and average margin. In: Kearns, M.J., Solla, S.A., Cohn, D.A. (eds.) Advances in neural information processing systems 11, p. 225–231. MIT Press, Cambridge (1999)
6. Khardon, R., Roth, D., Servedio, R.A.: Efficiency versus convergence of boolean kernels for on-line learning algorithms. J. Artif. Intell. Res. **24**, 341–356 (2005)
7. Kivinen, J., Warmuth, M.K.: Exponentiated gradient versus gradient descent for linear predictors. Inf. Comp. **132**(1), 1–64 (1997)
8. Klivans, A.R. Servedio, R.A.: Toward attribute efficient learning of decision lists and parities. J. Mach. Learn. Res. **7**(Apr), 587–602 (2006)
9. Littlestone, N.: Learning quickly when irrelevant attributes abound: A new linear threshold algorithm. Mach. Learn. **2**(4), 285–318 (1988)
10. Littlestone, N., Warmuth, M.K.: The weighted majority algorithm. Inf. Comp. **108**(2), 212–261 (1994)
11. Martin, R.K., Sethares, W.A., Williamson, R.C., Johnson, Jr., C.R.: Exploiting sparsity in adaptive filters. IEEE Trans. Signal Process. **50**(8), 1883–1894 (2002)
12. Mossel, E., O'Donnell, R., Servedio, R.A.: Learning functions of k relevant variables. J. Comp. Syst. Sci. **69**(3), 421–434 (2004)
13. Ng, A.Y.: Feature selection, L_1 vs. L_2 regularization, and rotational invariance. In: Greiner, R., Schuurmans, D. (eds.) Proceedings of the 21st International Conference on Machine Learning, pp 615–622. The International Machine Learning Society, Princeton (2004)
14. Vovk, V.: Aggregating strategies. In: Fulk, M., Case, J. (eds.) Proceedings of the 3rd Annual Workshop on Computational Learning Theory, p. 371–383. Morgan Kaufmann, San Mateo (1990)

Automated Search Tree Generation

2004; Gramm, Guo, Hüffner, Niedermeier

FALK HÜFFNER
Department of Math and Computer Science, University of Jena, Jena, Germany

Keywords and Synonyms

Automated proofs of upper bounds on the running time of splitting algorithms

Problem Definition

This problem is concerned with the automated development and analysis of search tree algorithms. Search tree algorithms are a popular way to find optimal solutions to NP-complete problems.[1] The idea is to recursively solve several smaller instances in such a way that at least one branch is a yes-instance if and only if the original instance is. Typically, this is done by trying all possibilities to contribute to a solution certificate for a small part of the input, yielding a small local modification of the instance in each branch.

For example, consider the NP-complete CLUSTER EDITING problem: can a given graph be modified by adding or deleting up to k edges such that the resulting graph is a *cluster graph*, that is, a graph that is a disjoint union of cliques? To give a search tree algorithm for CLUSTER EDITING, one can use the fact that cluster graphs are exactly the graphs that do not contain a P_3 (a path of 3 vertices) as an induced subgraph. One can thus solve CLUSTER EDITING by finding a P_3 and splitting it into 3 branches: delete the first edge, delete the second edge, or add the missing edge. By this characterization, whenever there is no P_3 found, one already has a cluster graph. The original instance has a solution with k modifications if and only if at least one of the branches has a solution with $k-1$ modifications.

Analysis

For NP-complete problems, the running time of a search tree algorithm only depends on the size of the search tree up to a polynomial factor , which depends on the number of branches and the reduction in size of each branch. If the algorithm solves a problem of size s and calls itself recursively for problems of sizes $s-d_1, \ldots, s-d_i$, then $(d_1, \ldots, d_i)$ is called the *branching vector* of this recursion. It is known that the size of the search tree is then $O(\alpha^s)$, where the *branching number* α is the only positive real root of the *characteristic polynomial*

$$z^d - z^{d-d_1} - \cdots - z^{d-d_i} \,, \qquad (1)$$

where $d = \max\{d_1, \ldots, d_i\}$. For the simple CLUSTER EDITING search tree algorithm and the size measure k, the branching vector is $(1, 1, 1)$ and the branching number is 3, meaning that the running time is up to a polynomial factor $O(3^k)$.

[1] For ease of presentation, only decision problems are considered; adaption to optimization problems is straightforward.

Case Distinction

Often, one can obtain better running times by distinguishing a number of cases of instances, and giving a specialized branching for each case. The overall running time is then determined by the branching number of the worst case. Several publications obtain such algorithms by hand (e. g., a search tree of size $O(2.27^k)$ for CLUSTER EDITING [4]); the topic of this work is how to automate this. That is, the problem is the following:

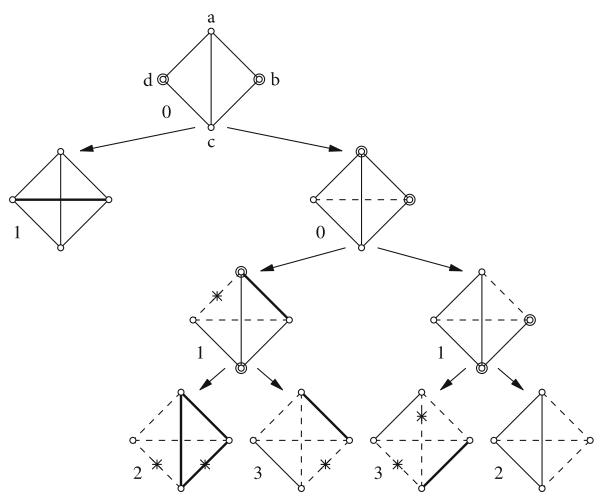
Problem 1 (Fast Search Tree Algorithm)
INPUT: *An NP-hard problem $\mathcal{P}$ and a size measure s(I) of an instance I of $\mathcal{P}$ where instances I with $s(I) = 0$ can be solved in polynomial time.*
OUTPUT: *A partition of the instance set of $\mathcal{P}$ into cases, and for each case a branching such that the maximum branching number over all branchings is as small as possible.*

Note that this problem definition is somewhat vague; in particular, to be useful, the case an instance belongs to must be recognizable quickly. It is also not clear whether an optimal search tree algorithm exists; conceivably, the branching number can be continuously reduced by increasingly complicated case distinctions.

Key Results

Gramm et al. [3] describe a method to obtain fast search tree algorithms for CLUSTER EDITING and related problems, where the size measure is the number of editing operations k. To get a case distinction, a number of subgraphs are enumerated such that each instance is known to contain at least one of these subgraphs. It is next described how to obtain a branching for a particular case.

A standard way of systematically obtaining specialized branchings for instance cases is to use a combination of *basic branching* and *data reduction rules*. Basic branching is typically a very simple branching technique, and data reduction rules replace an instance with a smaller, solution-equivalent instance in polynomial time. Applying this to CLUSTER EDITING first requires a small modification of the problem: one considers an *annotated* version, where an edge can be marked as *permanent* and a non-edge can be marked as *forbidden*. Any such annotated vertex pair cannot be edited anymore. For a pair of vertices, the basic branching then branches into two cases: permanent or forbidden (one of these options will require an editing operation). The reduction rules are: if two permanent edges are



Automated Search Tree Generation, Figure 1
Branching for a Cluster Editing case using only basic branching on vertex pairs (*double circles*), and applications of the reduction rules (*asterisks*). Permanent edges are marked *bold*, forbidden edges *dashed*. The *numbers* next to the subgraphs state the change of the problem size k. The branching vector is $(1, 2, 3, 3, 2)$, corresponding to a search tree size of $O(2.27^k)$

adjacent, the third edge of the triangle they induce must also be permanent; and if a permanent and a forbidden edge are adjacent, the third edge of the triangle they induce must be forbidden.

Figure 1 shows an example branching derived in this way.

Using a refined method of searching the space for all possible cases and to distinguish all branchings for a case, Gramm et al. [3] derive a number of search tree algorithms for graph modification problems.

Applications

Gramm et al. [3] apply the automated generation of search tree algorithms to several graph modification problems (see also Table 1). Further, Hüffner [5] demonstrates an application of Dominating Set on graphs with maximum degree 4, where the size measure is the size of the dominating set.

Fedin and Kulikov [2] examine variants of SAT; however, their framework is limited in that it only proves upper bounds for a fixed algorithm instead of generating algorithms.

Skjernaa [6] also presents results on variants of SAT. His framework does not require user-provided data reduction rules, but determines reductions automatically.

Automated Search Tree Generation, Table 1
Summary of search tree sizes where automation gave improvements. "Known" is the size of the best previously published "hand-made" search tree. For the satisfiability problems, m is the number of clauses and l is the length of the formula

Problem	Trivial	Known	New
Cluster Editing	3	2.27	1.92 [3]
Cluster Deletion	2	1.77	1.53 [3]
Cluster Vertex Deletion	3	2.27	2.26 [3]
Bounded Degree Dominating Set	4		3.71 [5]
X3SAT, size measure m	3	1.1939	1.1586 [6]
$(n, 3)$-MaxSAT, size measure m	2	1.341	1.2366 [2]
$(n, 3)$-MaxSAT, size measure l	2	1.1058	1.0983 [2]

Open Problems

The analysis of search tree algorithms can be much improved by describing the "size" of an instance by more than one variable, resulting in multivariate recurrences [1]. It is open to introduce this technique into an automation framework.

It has frequently been reported that better running time bounds obtained by distinguishing a large number of cases do not necessarily speed up, but in fact can slow down, a program. A careful investigation of the tradeoffs involved and a corresponding adaption of the automation frameworks is an open task.

Experimental Results

Gramm et al. [3] and Hüffner [5] report search tree sizes for several NP-complete problems. Further, Fedin and Kulikov [2] and Skjernaa [6] report on variants of satisfiability. Table 1 summarizes the results.

Cross References

► Vertex Cover Search Trees

Acknowledgments

Partially supported by the Deutsche Forschungsgemeinschaft, Emmy Noether research group PIAF (fixed-parameter algorithms), NI 369/4.

Recommended Reading

1. Eppstein, D.: Quasiconvex analysis of backtracking algorithms. In: Proc. 15th SODA, ACM/SIAM, pp. 788–797 (2004)
2. Fedin, S.S., Kulikov, A.S.: Automated proofs of upper bounds on the running time of splitting algorithms. J. Math. Sci. **134**, 2383–2391 (2006). Improved results at http://logic.pdmi.ras.ru/~kulikov/autoproofs.html

3. Gramm, J., Guo, J., Hüffner, F., Niedermeier, R.: Automated generation of search tree algorithms for hard graph modification problems. Algorithmica **39**, 321–347 (2004)
4. Gramm, J., Guo, J., Hüffner, F., Niedermeier, R.: Graph-modeled data clustering: Exact algorithms for clique generation. Theor. Comput. Syst. **38**, 373–392 (2005)
5. Hüffner, F.: Graph Modification Problems and Automated Search Tree Generation. Diplomarbeit, Wilhelm-Schickard-Institut für Informatik, Universität Tübingen (2003)
6. Skjernaa, B.: Exact Algorithms for Variants of Satisfiability and Colouring Problems. Ph. D. thesis, University of Aarhus, Department of Computer Science (2004)

B

Backtracking Based k-SAT Algorithms

2005; Paturi, Pudlák, Saks, Zane

RAMAMOHAN PATURI[1], PAVEL PUDLÁK[2],
MICHAEL SAKS[3], FRANCIS ZANE[4]
[1] Department of Computer Science and Engineering, University of California at San Diego, San Diego, CA, USA
[2] Mathematical Institute, Academy of Science of the Czech Republic, Prague, Czech Republic
[3] Department of Mathematics, Rutgers, State University of New Jersey, Piscataway, NJ, USA
[4] Bell Laboraties, Lucent Technologies, Murray Hill, NJ, USA

Problem Definition

Determination of the complexity of k-CNF satisfiability is a celebrated open problem: given a Boolean formula in conjunctive normal form with at most k literals per clause, find an assignment to the variables that satisfies each of the clauses or declare none exists. It is well-known that the decision problem of k–CNF satisfiability is NP-complete for $k \geq 3$. This entry is concerned with algorithms that significantly improve the worst case running time of the naive exhaustive search algorithm, which is poly$(n)2^n$ for a formula on n variables. Monien and Speckenmeyer [8] gave the first real improvement by giving a simple algorithm whose running time is $O(2^{(1-\varepsilon_k)n})$, with $\varepsilon_k > 0$ for all k. In a sequence of results [1,3,5,6,7,9,10,11,12], algorithms with increasingly better running times (larger values of ε_k) have been proposed and analyzed.

These algorithms usually follow one of two lines of attack to find a satisfying solution. Backtrack search algorithms make up one class of algorithms. These algorithms were originally proposed by Davis, Logemann and Loveland [4] and are sometimes called Davis–Putnam procedures. Such algorithms search for a satisfying assignment by assigning values to variables one by one (in some order), backtracking if a clause is made false. The other class of algorithms is based on local searches (the first guaranteed performance results were obtained by Schöning [12]). One starts with a randomly (or strategically) selected assignment, and searches locally for a satisfying assignment guided by the unsatisfied clauses.

This entry presents **ResolveSat**, a randomized algorithm for k-CNF satisfiability which achieves some of the best known upper bounds. **ResolveSat** is based on an earlier algorithm of Paturi, Pudlák and Zane [10], which is essentially a backtrack search algorithm where the variables are examined in a randomly chosen order. An analysis of the algorithm is based on the observation that as long as the formula has a satisfying assignment which is isolated from other satisfying assignments, a third of the variables are expected to occur as unit clauses as the variables are assigned in a random order. Thus, the algorithm needs to correctly guess the values of at most 2/3 of the variables. This analysis is extended to the general case by observing that there either exists an isolated satisfying assignment, or there are many solutions so the probability of guessing one correctly is sufficiently high.

ResolveSat combines these ideas with resolution to obtain significantly improved bounds [9]. In fact, **ResolveSat** obtains the best known upper bounds for k-CNF satisfiability for all $k \geq 5$. For $k = 3$ and 4, Iwama and Takami [6] obtained the best known upper bound with their randomized algorithm which combines the ideas from Schöning's local search algorithm and **ResolveSat**. Furthermore, for the promise problem of unique k-CNF satisfiability whose instances are conjectured to be among the hardest instances of k-CNF satisfiability [2], **ResolveSat** holds the best record for all $k \geq 3$. Bounds obtained by **ResolveSat** for unique k-SAT and k-SAT, for $k = 3, 4, 5, 6$ are shown in Table 1. Here, these bounds are compared with those of of Schöning [12], subsequently improved results based on local search [1,5,11], and the most recent improvements due to Iwama and Takami [6]. The upper bounds obtained by these algorithms are expressed in the form $2^{cn-o(n)}$ and the numbers in the table represent the exponent c. This comparison focuses only on the best bounds irrespective of the type of the algorithm (randomized versus deterministic).

Notation In this entry, a CNF boolean formula $F(x_1, x_2, \ldots, x_n)$ is viewed as both a boolean function and a set of clauses. A boolean formula F is a k-CNF if all the clauses have size at most k. For a clause C, write $var(C)$ for the set of variables appearing in C. If $v \in var(C)$, the *orientation* of v is positive if the literal v is in C and is negative if $\bar{v}$ is in C. Recall that if F is a CNF boolean formula on variables $(x_1, x_2, \ldots, x_n)$ and a is a partial assignment of the variables, the *restriction* of F by a is defined to be the formula $F' = F\lceil_a$ on the set of variables that are not set by a, obtained by treating each clause C of F as follows: if C is set to 1 by a then delete C, and otherwise replace C by the clause C' obtained by deleting any literals of C that are set to 0 by a. Finally, a *unit clause* is a clause that contains exactly one literal.

Key Results

ResolveSat Algorithm

The **ResolveSat** algorithm is very simple. Given a k-CNF formula, it first generates clauses that can be obtained by resolution without exceeding a certain clause length. Then it takes a random order of variables and gradually assigns values to them in this order. If the currently considered variable occurs in a unit clause, it is assigned the only value that satisfies the clause. If it occurs in contradictory unit clauses, the algorithm starts over. At each step, the algorithm also checks if the formula is satisfied. If the formula is satisfied, then the input is accepted. This subroutine is repeated until either a satisfying assignment is found or a given time limit is exceeded.

The **ResolveSat** algorithm uses the following subroutine, which takes an arbitrary assignment y, a CNF formula F, and a permutation π as input, and produces an assignment u. The assignment u is obtained by considering the variables of y in the order given by π and modifying their values in an attempt to satisfy F.

```
Function **Modify**(CNF formula $G(x_1, x_2, \ldots, x_n)$, permutation $\pi$ of $\{1, 2, \ldots, n\}$, assignment $y$) $\longrightarrow$ (assignment $u$)
  $G_0 = G$.
  **for** $i = 1$ **to** $n$
    **if** $G_{i-1}$ contains the unit clause $x_{\pi(i)}$
      **then** $u_{\pi(i)} = 1$
    **else if** $G_{i-1}$ contains the unit clause $\bar{x}_{\pi(i)}$
      **then** $u_{\pi(i)} = 0$
    **else** $u_{\pi(i)} = y_{\pi(i)}$
    $G_i = G_{i-1}\lceil_{x_{\pi(i)}=u_{\pi(i)}}$
  **end** /* end for loop */
  **return** $u$;
```

The algorithm **Search** is obtained by running **Modify**(G, π, y) on many pairs (π, y), where π is a random permutation and y is a random assignment.

```
**Search**(CNF-formula $F$, integer $I$)
  **repeat** $I$ times
    $\pi$ = uniformly random permutation of $1, \ldots, n$
    $y$ = uniformly random vector $\in \{0, 1\}^n$
    $u$ = **Modify**$(F, \pi, y)$;
    **if** $u$ satisfies $F$
      **then** output($u$); **exit**;
  **end**/* end repeat loop */
  output('Unsatisfiable');
```

The **ResolveSat** algorithm is obtained by combining **Search** with a preprocessing step consisting of *bounded resolution*. For the clauses C_1 and C_2, C_1 and C_2 *conflict* on variable v if one of them contains v and the other contains $\bar{v}$. C_1 and C_2 is a *resolvable pair* if they conflict on exactly one variable v. For such a pair, their *resolvent*, denoted $R(C_1, C_2)$, is the clause $C = D_1 \vee D_2$ where D_1 and D_2 are obtained by deleting v and $\bar{v}$ from C_1 and C_2. It is easy to see that any assignment satisfying C_1 and C_2 also satisfies C. Hence, if F is a satisfiable CNF formula containing the resolvable pair C_1, C_2 then the formula $F' = F \wedge R(C_1, C_2)$ has the same satisfying assignments as F. The resolvable pair C_1, C_2 is *s-bounded* if $|C_1|, |C_2| \leq s$ and $|R(C_1, C_2)| \leq s$. The following subroutine extends a formula F to a formula F_s by applying as many steps of s-bounded resolution as possible.

```
**Resolve**(CNF Formula $F$, integer $s$)
  $F_s = F$.
  **while** $F_s$ has an $s$-bounded resolvable pair $C_1, C_2$
      with $R(C_1, C_2) \notin F_s$
    $F_s = F_s \wedge R(C_1, C_2)$.
  **return** $(F_s)$.
```

The algorithm for k-SAT is the following simple combination of **Resolve** and **Search**:

```
**ResolveSat**(CNF-formula $F$, integer $s$, positive integer $I$)
  $F_s$ = **Resolve**$(F, s)$.
  **Search**$(F_s, I)$.
```

Backtracking Based *k*-SAT Algorithms, Table 1

This table shows the exponent *c* in the bound $2^{cn-o(n)}$ for the unique *k*-SAT and *k*-SAT from the ResolveSat algorithm, the bounds for *k*-SAT from Schöning's algorithm [12], its improved versions for 3-SAT [1,5,11], and the hybrid version of [6]

k	unique *k*-SAT [9]	*k*-SAT [9]	*k*-SAT [12]	*k*-SAT [1,5,11]	*k*-SAT [6]
3	0.386 …	0.521 …	0.415 …	0.409 …	0.404 …
4	0.554 …	0.562 …	0.584 …		0.559 …
5	0.650 …		0.678 …		
6	0.711 …		0.736 …		

Analysis of ResolveSat

The running time of **ResolveSat**(F, s, I) can be bounded as follows. **Resolve**(F, s) adds at most $O(n^s)$ clauses to F by comparing pairs of clauses, so a naive implementation runs in time $n^{3s}\operatorname{poly}(n)$ (this time bound can be improved, but this will not affect the asymptotics of the main results). **Search**(F_s, I) runs in time $I(|F| + n^s)\operatorname{poly}(n)$. Hence the overall running time of **ResolveSat**(F, s, I) is crudely bounded from above by $(n^{3s} + I(|F| + n^s))\operatorname{poly}(n)$. If $s = O(n/\log n)$, the overall running time can be bounded by $I|F|2^{O(n)}$ since $n^s = 2^{O(n)}$. It will be sufficient to choose s either to be some large constant or to be a *slowly growing* function of n. That is, $s(n)$ tends to infinity with n but is $O(\log n)$.

The algorithm **Search**(F, I) always answers "unsatisfiable" if F is unsatisfiable. Thus the only problem is to place an upper bound on the error probability in the case that F is satisfiable. Define $\tau(F)$ to be the probability that **Modify**(F, π, y) finds some satisfying assignment. Then for a satisfiable F the error probability of **Search**(F, I) is equal to $(1 - \tau(F))^I \le e^{-I\tau(F)}$, which is at most e^{-n} provided that $I \ge n/\tau(F)$. Hence, it suffices to give good upper bounds on $\tau(F)$.

Complexity analysis of **ResolveSat** requires certain constants μ_k for $k \ge 2$:

$$\mu_k = \sum_{j=1}^{\infty} \frac{1}{j(j + \frac{1}{k-1})} .$$

It is straightforward to show that $\mu_3 = 4 - 4\ln 2 > 1.226$ using Taylor's series expansion of $\ln 2$. Using standard facts, it is easy to show that μ_k is an increasing function of k with the limit $\sum_{j=1}^{\infty}(1/j^2) = (\pi^2/6) = 1.644\ldots$

The results on the algorithm **ResolveSat** are summarized in the following three theorems.

Theorem 1 ***(i)*** *Let $k \ge 5$, and let $s(n)$ be a function going to infinity. Then for any satisfiable k-CNF formula F on n variables,*

$$\tau(F_s) \ge 2^{-(1-\frac{\mu_k}{k-1})n-o(n)} .$$

Hence, ***ResolveSat***(F, s, I) *with* $I = 2^{(1-\mu_k/(k-1))n+O(n)}$ *has error probability $O(1)$ and running time $2^{(1-\mu_k/(k-1))n+O(n)}$ on any satisfiable k-CNF formula, provided that $s(n)$ goes to infinity sufficiently slowly.*

(ii) *For $k \ge 3$, the same bounds are obtained provided that F is uniquely satisfiable.*

Theorem 1 is proved by first considering the uniquely satisfiable case and then relating the general case to the uniquely satisfiable case. When $k \ge 5$, the analysis reveals that the asymptotics of the general case is no worse than that of the uniquely satisfiable case. When $k = 3$ or $k = 4$, it gives somewhat worse bounds for the general case than for the uniquely satisfiable case.

Theorem 2 *Let $s = s(n)$ be a slowly growing function. For any satisfiable n-variable 3-CNF formula, $\tau(F_s) \ge 2^{-0.521n}$ and so* ***ResolveSat***(F, s, I) *with $I = n2^{0.521n}$ has error probability $O(1)$ and running time $2^{0.521n+O(n)}$.*

Theorem 3 *Let $s = s(n)$ be a slowly growing function. For any satisfiable n-variable 4-CNF formula, $\tau(F_s) \ge 2^{-0.5625n}$, and so* ***ResolveSat***(F, s, I) *with $I = n2^{0.5625n}$ has error probability $O(1)$ and running time $2^{0.5625n+O(n)}$.*

Applications

Various heuristics have been employed to produce implementations of 3-CNF satisfiability algorithms which are considerably more efficient than exhaustive search algorithms. The **ResolveSat** algorithm and its analysis provide a rigorous explanation for this efficiency and identify the structural parameters (for example, the width of clauses and the number of solutions), influencing the complexity.

Open Problems

The gap between the bounds for the general case and the uniquely satisfiable case when $k \in \{3, 4\}$ is due to a weakness in analysis, and it is conjectured that the asymptotic bounds for the uniquely satisfiable case hold in general for all k. If true, the conjecture would imply that **ResolveSat** is also faster than any other known algorithm in the $k = 3$ case.

Another interesting problem is to better understand the connection between the number of satisfying assignments and the complexity of finding a satisfying assignment [2]. A strong conjecture is that satisfiability for formulas with many satisfying assignments is strictly easier than for formulas with fewer solutions.

Finally, an important open problem is to design an improved k-SAT algorithm which runs faster than the bounds presented in here for the unique k-SAT case.

Cross References

- ► Local Search Algorithms for kSAT
- ► Maximum Two-Satisfiability
- ► Parameterized SAT
- ► Thresholds of Random k-SAT

Recommended Reading

1. Baumer, S., Schuler, R.: Improving a Probabilistic 3-SAT Algorithm by Dynamic Search and Independent Clause Pairs. In: SAT 2003, pp. 150–161
2. Calabro, C., Impagliazzo, R., Kabanets, V., Paturi, R.: The Complexity of Unique k-SAT: An Isolation Lemma for k-CNFs. In: Proceedings of the Eighteenth IEEE Conference on Computational Complexity, 2003
3. Dantsin, E., Goerdt, A., Hirsch, E.A., Kannan, R., Kleinberg, J., Papadimitriou, C., Raghavan, P., Schöning, U.: A deterministic $(2 - \frac{2}{k+1})^n$ algorithm for k-SAT based on local search. Theor. Comp. Sci. **289**(1), 69–83 (2002)
4. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem proving. Commun. ACM **5**, 394–397 (1962)
5. Hofmeister, T., Schöning, U., Schuler, R., Watanabe, O.: A probabilistic 3–SAT algorithm further improved. In: STACS 2002. LNCS, vol. 2285, pp. 192–202. Springer, Berlin (2002)
6. Iwama, K., Tamaki, S.: Improved upper bounds for 3-SAT. In: Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms, 2004, pp. 328–329
7. Kullmann, O.: New methods for 3-SAT decision and worst-case analysis. Theor. Comp. Sci. **223**(1–2), 1–72 (1999)
8. Monien, B., Speckenmeyer, E.: Solving Satisfiability In Less Than 2^n Steps. Discret. Appl. Math. **10**, 287–295 (1985)
9. Paturi, R., Pudlák, P., Saks, M., Zane, F.: An Improved Exponential-time Algorithm for k-SAT. J. ACM **52**(3), 337–364 (2005) (An earlier version presented in Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science, 1998, pp. 628–637)
10. Paturi, R., Pudlák, P., Zane, F.: Satisfiability Coding Lemma. In: Proceedings of the 38th Annual IEEE Symposium on Foundations of Computer Science, 1997, pp. 566–574. Chicago J. Theor. Comput. Sci. (1999), http://cjtcs.cs.uchicago.edu/
11. Rolf, D.: 3-SAT $\in$ *RTIME*(1.32971^n). In: ECCC TR03-054, 2003
12. Schöning, U.: A probabilistic algorithm for k-SAT based on limited local search and restart. Algorithmica **32**, 615–623 (2002) (An earlier version appeared in 40th Annual Symposium on Foundations of Computer Science (FOCS '99), pp. 410–414)

Best Response Algorithms for Selfish Routing

2005; Fotakis, Kontogiannis, Spirakis

PAUL SPIRAKIS
Computer Engineering and Informatics, Research and Academic Computer Technology Institute, Patras University, Patras, Greece

Keywords and Synonyms

Atomic selfish flows

Problem Definition

A setting is assumed in which n selfish users compete for routing their loads in a network. The network is an $s-t$ directed graph with a single source vertex s and a single destination vertex t. The users are ordered sequentially. It is assumed that each user plays after the user before her in the ordering, and the desired end result is a Pure Nash Equilibrium (PNE for short). It is assumed that, when a user plays (i. e. when she selects an $s-t$ path to route her load), the play is a best response (i. e. minimum delay), given the paths and loads of users currently in the net. The problem then is to find the class of directed graphs for which such an ordering exists so that the implied sequence of best responses leads indeed to a Pure Nash Equilibrium.

The Model

A *network congestion game* is a tuple $((w_i)_{i \in N}, G, (d_e)_{e \in E})$ where $N = \{1, \dots, n\}$ is the set of users where user i controls w_i units of traffic demand. In *unweighted* congestion games $w_i = 1$ for $i = 1, \dots, n$. $G(V,E)$ is a directed graph representing the communications network and d_e is the latency function associated with edge $e \in E$. It is assumed that the d_e's are non-negative and non-decreasing functions of the edge loads. The edges are called *identical* if $d_e(x) = x, \ \forall e \in E$. The model is further restricted to single-commodity network congestion games, where G has a single source s and destination t and the set of users' strategies is the set of $s-t$ paths, denoted P. Without loss of generality it is assumed that G is connected and that every vertex of G lies on a directed $s-t$ path.

A vector $P = (p_1, \dots, p_n)$ consisting of an $s-t$ path p_i for each user i is a *pure strategies profile*. Let $l_e(P) = \sum_{i: e \in p_i} w_i$ be the load of edge e in P. The authors define *the cost* $\lambda^i_p(P)$ for user i routing her demand on path p in the profile P to be

$$\lambda_p^i(P) = \sum_{e \in p \cap p_i} d_e\left(l_e(P)\right) + \sum_{e \in p \smallsetminus p_i} d_e\left(l_e(P) + w_i\right) .$$

The cost $\lambda^i(P)$ of user i in P is just $\lambda^i_{p_i}(P)$, i. e. the total delay along her path.

A pure strategies profile P is a Pure Nash Equilibrium (PNE) iff no user can reduce her total delay by *unilaterally deviating* i. e. by selecting another $s - t$ path for her load, while all other users keep their paths.

Best Response

Let p_i be the path of user i and $P^i = \left(p_1, \ldots, p_i\right)$ be the pure strategies profile for users $1, \ldots, i$. Then the *best response* of user $i+1$ is a path p_{i+1} so that

$$p_{i+1} = avg \min_{p \in P^i} \left\{ \sum_{e \in p} \left(d_e \left(l_e \left(P^i \right) + w_{i+1} \right) \right) \right\} .$$

Flows and Common Best Response

A (feasible) flow on the set P of $s - t$ paths of G is a function $f\colon P \to \Re_{\geq 0}$ so that

$$\sum_{p \in P} f_p = \sum_{i=1}^{n} w_i .$$

The single-commodity network congestion game $((w_i)_{i \in N}, G, (d_e)_{e \in E})$ has the Common Best Response property if for every initial flow f (not necessarily feasible), all users have the same set of best responses with respect to f. That is, if a path p is a best response with respect to f for some user, then for all users j and all paths p'

$$\sum_{e \in p'} d_e\left(f_e + w_j\right) \geq \sum_{e \in p} d_e\left(f_e + w_j\right) .$$

Furthermore, every segment π of a best response path p is a best response for routing the demand of any user between π's endpoints. It is allowed here that some users may already have contributed to the initial flow f.

Layered and Series-Parallel Graphs

A directed (multi)graph $G(V, E)$ with a distinguished source s and destination t is *layered* iff all directed $s - t$ paths have exactly the same length and each vertex lies on some directed $s - t$ path.

A multigraph is *series-parallel* with *terminals* (s, t) if

1. it is a single edge (s, t) or
2. it is obtained from two series-parallel graphs G_1, G_2 with terminals (s_1, t_1) and (s_2, t_2) by connecting them either in *series* or in *parallel*. In a series connection, t_1 is identified with s_2 and s_1 becomes s and t_2 becomes t. In a parallel connection, $s_1 = s_2 = s$ and $t_1 = t_2 = t$.

Key Results

The Greedy Best Response Algorithm (GBR)

GBR considers the users one-by-one in *non-increasing* order of weight (i. e. $w_1 \geq w_2 \geq \cdots \geq w_n$). Each user adopts her best response strategy on the set of (already adopted in the net) best responses of previous users. No user can change her strategy in the future. Formally, GBR *succeeds* if the eventual profile P is a Pure Nash Equilibrium (PNE).

The Characterization

In [3] it is shown:

Theorem 1 *If G is an $(s - t)$ series-parallel graph and the game $((w_i)_{i \in N}, G, (d_e)_{e \in E})$ has the common best response property, then GBR succeeds.*

Theorem 2 *A weighted single-commodity network congestion game in a layered network with identical edges has the common best response property for any set of user weights.*

Theorem 3 *For any single-commodity network congestion game in series-parallel networks, GBR succeeds if*
1. *The users are identical (if $w_i = 1$ for all i) and the edge-delays are arbitrary but non-decreasing or*
2. *The graph is layered and the edges are identical (for arbitrary user weights)*

Theorem 4 *If the network consists of bunches of parallel-links connected in series, then a PNE is obtained by applying GBR to each bunch.*

Theorem 5
1. *If the network is not series-parallel then there exist games where GBR fails, even for 2 identical users and identical edges.*
2. *If the network does not have the common best response property (and is not a sequence of parallel links graphs connected in series) then there exist games where GBR fails, even for 2-layered series-parallel graphs.*

Examples of such games are provided in [3].

Applications

GBR has a natural distributed implementation based on a leader election algorithm. Each player is now represented by a process. It is assumed that processes know the network and the edge latency functions. The existence of a message passing subsystem and an underlying synchronization mechanism (e. g. logical timestamps) is assumed, that allows a distributed protocol to proceed in logical rounds.

Initially all processes are active. In each round they run a leader election algorithm and determine the process of largest weight (among the active ones). This process routes its demand on its best response path, announces its strategy to all active processes, and becomes passive. Notice that each process can compute its best response locally.

Open Problems

What is the class of networks where (identical) users can achieve a PNE by a k-round repetition of a best responses sequence? What happens to weighted users? In general, how the network topology affects best response sequences? Such open problems are a subject of current research.

Cross References

► General Equilibrium

Recommended Reading

1. Awerbuch, B., Azar, Y., Epstein, A.: The price of Routing Unsplittable Flows. In: Proc. ACM Symposium on Theory of Computing (STOC) 2005, pp. 57-66. ACM, New York (2005)
2. Duffin, R.J.: Topology of Series-Parallel Networks. J. Math. Anal. Appl. **10**, 303–318 (1965)
3. Fotakis, D., Kontogiannis, S., Spirakis, P.: Symmetry in Network Congestion Games: Pure Equilibria and Anarchy Cost. In: Proc. of the 3rd Workshop of Approximate and On-line Algorithms (WAOA 2005). Lecture Notes in Computer Science (LNCS), vol. 3879, pp. 161–175. Springer, Berlin Heidelberg (2006)
4. Fotakis, D., Kontogiannis, S., Spirakis, P.: Selfish Unsplittable Flows. J. Theor. Comput. Sci. **348**, 226–239 (2005)
5. Libman, L., Orda, A.: Atomic Resource Sharing in Noncooperative Networks. Telecommun. Syst. **17**(4), 385-409 (2001)

Bidimensionality

2004; Demaine, Fomin, Hajiaghayi, Thilikos

ERIK D. DEMAINE[1], MOHAMMADTAGHI HAJIAGHAYI[2]

[1] Computer Science and Artifical Intelligence Laboratory, MIT, Cambridge, MA, USA

[2] Department of Computer Science, University of Pittsburgh, Pittsburgh, PA, USA

Problem Definition

The theory of bidimensionality provides general techniques for designing efficient fixed-parameter algorithms and approximation algorithms for a broad range of NP-hard graph problems in a broad range of graphs. This theory applies to graph problems that are "bidimensional" in the sense that (1) the solution value for the $k \times k$ grid graph and similar graphs grows with k, typically as $\Omega(k^2)$, and (2) the solution value goes down when contracting edges and optionally when deleting edges in the graph. Many problems are bidimensional; a few classic examples are vertex cover, dominating set, and feedback vertex set.

Graph Classes

Results about bidimensional problems have been developed for increasingly general families of graphs, all generalizing planar graphs.

The first two classes of graphs relate to embeddings on surfaces. A graph is *planar* if it can be drawn in the plane (or the sphere) without crossings. A graph has *(Euler) genus* at most g if it can be drawn in a surface of Euler characteristic g. A class of graphs has *bounded genus* if every graph in the class has genus at most g for a fixed g.

The next three classes of graphs relate to excluding minors. Given an edge $e = \{v, w\}$ in a graph G, the *contraction* of e in G is the result of identifying vertices v and w in G and removing all loops and duplicate edges. A graph H obtained by a sequence of such edge contractions starting from G is said to be a *contraction* of G. A graph H is a *minor* of G if H is a subgraph of some contraction of G. A graph class C is *minor-closed* if any minor of any graph in C is also a member of C. A minor-closed graph class C is *H-minor-free* if $H \notin C$. More generally, the term "H-minor-free" refers to any minor-closed graph class that excludes some fixed graph H. A *single-crossing graph* is a minor of a graph that can be drawn in the plane with at most one pair of edges crossing. A minor-closed graph class is *single-crossing-minor-free* if it excludes a fixed single-crossing graph. An *apex graph* is a graph in which the removal of some vertex leaves a planar graph. A graph class is *apex-minor-free* if it excludes some fixed apex graph.

Bidimensional Parameters

Although implicitly hinted at in [2,5,10,11], the first use of the term "bidimensional" was in [3].

First, "parameters" are an alternative view on optimization problems. A *parameter* P is a function mapping graphs to nonnegative integers. The *decision problem associated with* P asks, for a given graph G and nonnegative integer k, whether $P(G) \le k$. Many optimization problems can be phrased as such a decision problem about a graph parameter P.

Now, a parameter is *$g(r)$-bidimensional* (or just *bidimensional*) if it is at least $g(r)$ in an $r \times r$ "grid-like graph" and if the parameter does not increase when taking either minors $g(r)$(*-minor-bidimensional*) or contractions ($g(r)$*-contraction-bidimensional*). The exact definition of "grid-like graph" depends on the class of graphs allowed and whether one considers minor- or contraction-bidimensionality. For minor-bidimensionality and for any H-minor-free graph class, the notion of a "grid-like graph" is defined to be the *$r \times r$ grid*, i. e., the planar graph with r^2 vertices arranged on a square grid and with edges connecting horizontally and vertically adjacent vertices. For contraction-bidimensionality, the notion of a "grid-like graph" is as follows:

1. For planar graphs and single-crossing-minor-free graphs, a "grid-like graph" is an $r \times r$ grid partially triangulated by additional edges that preserve planarity.
2. For bounded-genus graphs, a "grid-like graph" is such a partially triangulated $r \times r$ grid with up to genus(G) additional edges ("handles").
3. For apex-minor-free graphs, a "grid-like graph" is an $r \times r$ grid augmented with additional edges such that each vertex is incident to $O(1)$ edges to nonboundary vertices of the grid. (Here $O(1)$ depends on the excluded apex graph.)

Contraction-bidimensionality is so far undefined for H-minor-free graphs (or general graphs).

Examples of bidimensional parameters include the number of vertices, the diameter, and the size of various structures such as feedback vertex set, vertex cover, minimum maximal matching, face cover, a series of vertex-removal parameters, dominating set, edge dominating set, R-dominating set, connected dominating set, connected edge dominating set, connected R-dominating set, unweighted TSP tour (a walk in the graph visiting all vertices), and chordal completion (fill-in). For example, feedback vertex set is $\Omega(r^2)$-minor-bidimensional (and thus also contraction-bidimensional) because (1) deleting or contracting an edge preserves existing feedback vertex sets, and (2) any vertex in the feedback vertex set destroys at most four squares in the $r \times r$ grid, and there are $(r-1)^2$ squares, so any feedback vertex set must have $\Omega(r^2)$ vertices. See [1,3] for arguments of either contraction- or minor-bidimensionality for the other parameters.

Key Results

Bidimensionality builds on the seminal Graph Minor Theory of Robertson and Seymour, by extending some mathematical results and building new algorithmic tools. The foundation for several results in bidimensionality are the following two combinatorial results. The first relates any bidimensional parameter to treewidth, while the second relates treewidth to grid minors.

Theorem 1 ([1,8]) *If the parameter P is $g(r)$-bidimensional, then for every graph G in the family associated with the parameter P, $tw(G) = O(g^{-1}(P(G)))$. In particular, if $g(r) = \Theta(r^2)$, then the bound becomes $tw(G) = O(\sqrt{P(G)})$.*

Theorem 2 ([8]) *For any fixed graph H, every H-minor-free graph of treewidth w has an $\Omega(w) \times \Omega(w)$ grid as a minor.*

The two major algorithmic results in bidimensionality are general subexponential fixed-parameter algorithm, and general polynomial-time approximation scheme (PTASs):

Theorem 3 ([1,8]) *Consider a $g(r)$-bidimensional parameter P that can be computed on a graph G in $h(w)n^{O(1)}$ time given a tree decomposition of G of width at most w. Then there is an algorithm computing P on any graph G in P's corresponding graph class, with running time $[h(O(g^{-1}(k))) + 2^{O(g^{-1}(k))}]n^{O(1)}$. In particular, if $g(r) = \Theta(r^2)$ and $h(w) = 2^{o(w^2)}$, then this running time is subexponential in k.*

Theorem 4 ([7]) *Consider a bidimensional problem satisfying the "separation property" defined in [4,7]. Suppose that the problem can be solved on a graph G with n vertices in $f(n, tw(G))$ time. Suppose also that the problem can be approximated within a factor of α in $g(n)$ time. For contraction-bidimensional problems, suppose further that both of these algorithms also apply to the "generalized form" of the problem defined in [4,7]. Then there is a $(1+\epsilon)$-approximation algorithm whose running time is $O(nf(n, O(\alpha^2/\epsilon)) + n^3 g(n))$ for the corresponding graph class of the bidimensional problem.*

Applications

The theorems above have many combinatorial and algorithmic applications.

Applying the parameter-treewidth bound of Theorem 1 to the parameter of the number of vertices in the graph proves that every H-minor-free graph on n vertices has treewidth $O(\sqrt{n})$, thus (re)proving the separator theorem for H-minor-free graphs. Applying the parameter-treewidth bound of Theorem 1 to the parameter of the diameter of the graph proves a stronger form of Eppstein's diameter-treewidth relation for apex-minor-free graphs. (Further work shows how to further strengthen the diameter-treewidth relation to linear [6].) The treewidth-grid relation of Theorem 2 can be used to bound the gap between half-integral multicommodity flow and fractional multicommodity flow in H-minor-free graphs. It also yields an $O(1)$-approximation for treewidth in H-minor-free graphs. The subexponential fixed-parameter algorithms of Theorem 3 subsume or strengthen all previous such results. These results can also be generalized to obtain fixed-parameter algorithms in arbitrary graphs. The PTASs of Theorem 4 in particular establish the first PTASs for connected dominating set and feedback vertex set even for planar graphs. For details of all of these results, see [4].

Open Problems

Several combinatorial and algorithmic open problems remain in the theory of bidimensionality and related concepts.

Can the grid-minor theorem for H-minor-free graphs, Theorem 2, be generalized to arbitrary graphs with a polynomial relation between treewidth and the largest grid minor? (The best relation so far is exponential.) Such polynomial generalizations have been obtained for the cases of "map graphs" and "power graphs" [9]. Good grid-treewidth bounds have applications to minor-bidimensional problems.

Can the algorithmic results (Theorem 3 and Theorem 4) be generalized to solve contraction-bidimensional problems beyond apex-minor-free graphs? It is known that the basis for these results, Theorem 1, does not generalize [1]. Nonetheless, Theorem 3 has been generalized for one specific contraction-bidimensional problem, dominating set [3].

Can the polynomial-time approximation schemes of Theorem 4 be generalized to more general algorithmic problems that do not correspond directly to bidimensional parameters? One general family of such problems arises when adding weights to vertices and/or edges, and the goal is e. g. to find the minimum-weight dominating set. Another family of such problems arises when placing constraints (e. g., on coverage or domination) only on subsets of vertices and/or edges. Examples of such problems include Steiner tree and subset feedback vertex set.

For additional open problems and details about the problems above, see [4].

Cross References

► Approximation Schemes for Planar Graph Problems
► Branchwidth of Graphs
► Treewidth of Graphs

Recommended Reading

1. Demaine, E.D., Fomin, F.V., Hajiaghayi, M., Thilikos, D.M.: Bidimensional parameters and local treewidth. SIAM J. Discret. Math. **18**(3), 501–511 (2004)
2. Demaine, E.D., Fomin, F.V., Hajiaghayi, M., Thilikos, D.M.: Fixed-parameter algorithms for (k, r)-center in planar graphs and map graphs. ACM Trans. Algorithms **1**(1), 33–47 (2005)
3. Demaine, E.D., Fomin, F.V., Hajiaghayi, M., Thilikos, D.M.: Subexponential parametrized algorithms on graphs of bounded genus and H-minor-free graphs. J. ACM **52**(6), 866–893 (2005)
4. Demaine, E.D., Hajiaghayi, M.: The bidimensionality theory and its algorithmic applications. Comput. J. To appear
5. Demaine, E.D., Hajiaghayi, M.: Diameter and treewidth in minor-closed graph families, revisited. Algorithmica **40**(3), 211–215 (2004)
6. Demaine, E.D., Hajiaghayi, M.: Equivalence of local treewidth and linear local treewidth and its algorithmic applications. In: Proceedings of the 15th ACM-SIAM Symposium on Discrete Algorithms (SODA'04), January 2004, pp. 833–842 (2004)
7. Demaine, E.D., Hajiaghayi, M.: Bidimensionality: New connections between FPT algorithms and PTASs. In: Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2005), pp. 590–601. Vancouver, January (2005)
8. Demaine, E.D., Hajiaghayi, M.: Graphs excluding a fixed minor have grids as large as treewidth, with combinatorial and algorithmic applications through bidimensionality. In: Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2005), pp. 682–689. Vancouver, January (2005)
9. Demaine, E.D., Hajiaghayi, M., Kawarabayashi, K.: Algorithmic graph minor theory: Improved grid minor bounds and Wagner's contraction. In: Proceedings of the 17th Annual International Symposium on Algorithms and Computation, Calcutta, India, December 2006. Lecture Notes in Computer Science, vol. 4288, pp. 3–15 (2006)
10. Demaine, E.D., Hajiaghayi, M., Nishimura, N., Ragde, P., Thilikos, D.M.: Approximation algorithms for classes of graphs excluding single-crossing graphs as minors. J. Comput. Syst. Sci. **69**(2), 166–195 (2004)
11. Demaine, E.D., Hajiaghayi, M., Thilikos, D.M.: Exponential speedup of fixed-parameter algorithms for classes of graphs excluding single-crossing graphs as minors. Algorithmica **41**(4), 245–267 (2005)
12. Demaine, E.D., Hajiaghayi, M., Thilikos, D.M.: The bidimensional theory of bounded-genus graphs. SIAM J. Discret. Math. **20**(2), 357–371 (2006)

Binary Decision Graph

1986; Bryant

AMIT PRAKASH[1], ADNAN AZIZ[2]
[1] Microsoft, MSN, Redmond, WA, USA
[2] Department of Electrical and Computer Engineering, University of Texas, Austin, TX, USA

Keywords and Synonyms

BDDs; Binary decision diagrams

Problem Definition

Boolean Functions

The concept of a *Boolean function* – a function whose domain is $\{0,1\}^n$ and range is $\{0,1\}$ – is central to computing. Boolean functions are used in foundational studies of complexity [7,9], as well as the design and analysis of logic circuits [4,13]. A Boolean function can be represented using a *truth table* – an enumeration of the values taken by the function on each element of $\{0,1\}^n$. Since the truth table representation requires memory exponential in n, it is impractical for most applications. Consequently, there is a need for data structures and associated algorithms for efficiently representing and manipulating Boolean functions.

Boolean Circuits

Boolean functions can be represented in many ways. One natural representation is a *Boolean combinational circuit*, or circuit for short [6, Chapter 34]. A circuit consists of *Boolean combinational elements* connected by *wires*. The Boolean combinational elements are *gates* and *primary inputs*. Gates come in three types: NOT, AND, and OR. The NOT gate functions as follows: it takes a single Boolean-valued *input*, and produces a single Boolean-valued *output* which takes value 0 if the input is 1, and 1 if the input is 0. The AND gate takes two Boolean-valued inputs and produce a single output; the output is 1 if both inputs are 1, and 0 otherwise. The OR gate is similar to AND, except that its output is 1 if one or both inputs are 1, and 0 otherwise.

Circuits are required to be acyclic. The absence of cycles implies that a Boolean-assignment to the primary inputs can be unambiguously propagated through the gates in topological order. It follows that a circuit on n ordered primary inputs with a designated gate called the *primary output* corresponds to a Boolean function on $\{0,1\}^n$. Every Boolean function can be represented by a circuit, e. g., by building a circuit that mimics the truth table.

The circuit representation is very general – any decision problem that is computable in polynomial-time on a Turing machine can be computed by circuits polynomial in the instance size, and the circuits can be constructed efficiently from the Turing machine program [15]. However, the key analysis problems on circuits, namely satisfiability and equivalence, are NP-hard [7].

Boolean Formulas

A *Boolean formula* is defined recursively: a *Boolean variable* x_i is a *Boolean formula*, and if φ and ψ are Boolean formulas, then so are $(\neg\phi)$, $(\phi\wedge\psi)$, $(\phi\vee\psi)$, $(\phi\rightarrow\psi)$, and $(\phi\leftrightarrow\psi)$. The operators $\neg, \vee, \wedge, \rightarrow, \leftrightarrow$ are referred to as *connectives*; parentheses are often dropped for notational convenience. Boolean formulas also can be used to represent arbitrary Boolean functions; however, formula satisfiability and equivalence are also NP-hard. Boolean formulas are not as succinct as Boolean circuits: for example, the parity function has linear sized circuits, but formula representations of parity are super-polynomial. More precisely, $XOR_n\colon \{0,1\}^n \mapsto \{0,1\}$ is defined to take the value 1 on exactly those elements of $\{0,1\}^n$ which contain an odd number of 1s. Define the *size* of a formula to be the number of connectives appearing in it. Then for any sequence of formulas $\theta_1, \theta_2, \ldots$ such that θ_k represents XOR_k, the size of θ_k is $\Omega(k^c)$ for all $c \in Z^+$ [14, Chapters 11,12].

A *disjunct* is a Boolean formula in which $\wedge$ and $\neg$ are the only connectives, and $\neg$ is applied only to variables; for example, $x_1 \wedge \neg x_3 \wedge \neg x_5$ is a disjunct. A Boolean formula is said to be in *Disjunctive Normal Form* (DNF) if it is of the form $D_0 \vee D_1 \vee \cdots \vee D_{k-1}$, where each D_i is a disjunct. DNF formulas can represent arbitrary Boolean functions, e. g., by identifying each input on which the formula takes the value 1 with a disjunct. DNF formulas are useful in logic design, because it can be translated directly into a PLA implementation [4]. While satisfiability of DNF formulas is trivial, equivalence is NP-hard. In addition, given DNF formulas φ and ψ, the formulas $\neg\phi$ and $\phi\wedge\psi$ are not DNF formulas, and the translation of these formulas to DNF formulas representing the same function can lead to exponential growth in the size of the formula.

Shannon Trees

Let f be a Boolean function on domain $\{0,1\}^n$. Associate the n dimensions with variables $x_0, \ldots, x_{n-1}$. Then the *positive cofactor* of f with respect to x_i, denoted by f_{x_i}, is the function on domain $\{0,1\}^n$, which is defined by

$$\begin{aligned} &f_{x_i}(\alpha_0, \ldots, \alpha_{i-1}, a_i, \alpha_{i+1}, \ldots, \alpha_{n-1}) \\ &= f(\alpha_0, \ldots, \alpha_{i-1}, 1, \alpha_{i+1}, \ldots, \alpha_{n-1}) \,. \end{aligned}$$

The *negative cofactor* of f with respect to x_i, denoted by $f_{x_i'}$ is defined similarly, with 0 taking the place of 1 in the right-hand side.

Every Boolean function can be decomposed using Shannon's expansion theorem:

$$f(x_1, \ldots, x_n) = x_i \cdot f_{x_i} + x_i' \cdot f_{x_i'} \,.$$

This observation can be used to represent f by a *Shannon tree* – a full binary tree [6, Appendix B.5] of height n, where each path to a leaf node defines a complete assignment to the n variables that f is defined over, and the leaf node holds a 0 or a 1, based on the value f takes for the assignment.

The Shannon tree is not a particularly useful representation, since the height of the tree representing every Boolean function on $\{0,1\}^n$ is n, and the tree has 2^n leaves. The Shannon tree can be made smaller by merging isomorphic subtrees, and bypassing nodes which have identical children. At first glance the reduced Shannon tree representation is not particularly useful, since it entails creating the full binary tree in the first place. Furthermore, it is not clear how to efficiently perform computations on the reduced Shannon tree representation, such as equivalence checking or computing the conjunction of functions presented as reduced Shannon trees.

Bryant [5] recognized that adding the restriction that variables appear in fixed order from root to leaves greatly reduced the complexity of manipulating reduced Shannon trees. He referred to this representation as a Binary Decision Diagram (BDD).

Key Results

Definitions

Technically, a BDD is a directed acyclic graph (DAG), with a designated root, and at most two sinks – one labeled 0, the other labeled 1. Nonsink nodes are labeled with a variable. Each nonsink node has two outgoing edges – one labeled with a 1 leading to the *1-child*, the other is a 0, leading to the *0-child*. Variables must be ordered – that is if the variable label x_i appears before the label x_j on some path from the root to a sink, then the label x_j is precluded from appearing before x_i on any path from the root to a sink. Two nodes are *isomorphic* if both are equi-labeled sinks, or they are both nonsink nodes, with the same variable label, and their 0- and 1-children are isomorphic. For the DAG to be a valid BDD, it is required that there are no isomorphic nodes, and for no nodes are its 0- and 1-children the same.

A key result in the theory of BDDs is that given a fixed variable ordering, the representation is unique upto isomorphism, i. e., if F and G are both BDDs representing $f : \{0,1\}^n \mapsto \{0,1\}$ under the variable ordering $x_1 \prec x_2 \prec \cdots x_n$, then F and G are isomorphic.

The definition of isomorphism directly yields a recursive algorithm for checking isomorphism. However, the resulting complexity is exponential in the number of nodes – this is illustrated for example by checking the isomorphism of the BDD for the parity function against itself. On inspection, the exponential complexity arises from repeated checking of isomorphism between pairs of nodes – this naturally suggest dynamic programming. Caching isomorphism checks reduces the complexity of isomorphism checking to $O(|F| \cdot |G|)$, where $|B|$ denotes the number of nodes in the BDD B.

BDD Operations

Many logical operations can be implemented in polynomial time using BDDs: *bdd_and* which computes a BDD representing the logical AND of the functions represented by two BDDs, *bdd_or* and *bdd_not* which are defined similarly, and *bdd_compose* which takes a BDD representing a function f, a variable v, and a BDD representing a function g and returns the BDD for f where v is substituted by g are examples.

The example of *bdd_and* is instructive – it is based on the identity $f \cdot g = x \cdot (f_x \cdot g_x) + x\prime \cdot (f_{x\prime} \cdot g_{x\prime})$. The recursion can be implemented directly: the base cases are when either f or g are 0, and when one or both are 1. The recursion chooses the variable v labeling either the root of the BDD for f or g, depending on which is earlier in the variable ordering, and recursively computes BDDs for $f_v \cdot g_v$ and $f_{v'} \cdot g_{v'}$; these are merged if isomorphic. Given a BDD F for f, if v is the variable labeling the root of F, the BDDs for $f_{v'}$ and f_v respectively are simply the 0-child and 1-child of F's root.

The implementation of *bdd_and* as described has exponential complexity because of repeated subproblems arising. Dynamic programming again provides a solution – caching the intermediate results of *bdd_and* reduced the complexity to $O(|F| \cdot |G|)$.

Variable Ordering

All symmetric functions on $\{0,1\}^n$, have a BDD that is polynomial in n, independent of the variable ordering. Other useful functions such as comparators, multiplexers, adders, and subtracters can also be efficiently represented, if the variable ordering is selected correctly. Heuristics for ordering selection are presented in [1,2,11]. There are functions which do not have a polynomial-sized BDD under any variable ordering – the function representing the n-th bit of the output of a multiplier taking two n-bit unsigned integer inputs is an example [5]. Wegener [17] presents many more examples of the impact of variable ordering.

Applications

BDDs have been most commonly applied in the context of formal verification of digital hardware [8]. Digital hardware extends the notion of circuit described above by adding *state elements* which hold a Boolean value between updates, and are updated on a *clock* signal.

The gates comprising a design are often updated based on performance requirements; these changes typically are not supposed to change the logical functionality of the design. BDD-based approaches have been used for checking the equivalence of digital hardware designs [10].

BDDs have also been used for checking properties of digital hardware. A typical formulation is that a set of "good" states and a set of "initial" states are specified using Boolean formulas over the state elements; the property holds iff there is no sequence of inputs which leads a state in the initial state to a state not in the set of good states. Given a design with n registers, a set of states A in the design can be characterized by a formula φ_A over n Boolean variables: φ_A evaluates to true on an assignment to the variables iff the corresponding state is in A. The formula φ_A represents a Boolean function, and so BDDs can be used to represent sets of states. The key operation of computing the *image* of a set of states A, i. e., the set of states that can be reached on application of a single input from states in A, can also be implemented using BDDs [12].

BDDs have been used for *test generation*. One approach to test generation is to specify legal inputs using constraints, in essence Boolean formulas over the the primary input and state variables. Yuan et al. [18] have demonstrated that BDDs can be used to solve these constraints very efficiently.

Logic synthesis is the discipline of realizing hardware designs specified as logic equations using gates. Mapping equations to gates is straightforward; however, in practice a direct mapping leads to implementations that are not acceptable from a performance perspective, where performance is measured by gate area or timing delay. Manipulating logic equations in order to reduce area (e. g., through constant propagation, identifying common subexpressions, etc.), and delay (e. g., through propagating late arriving signals closer to the outputs), is conveniently done using BDDs.

Experimental Results

Bryant reported results on verifying two qualitatively distinct circuits for addition. He was able to verify on a VAX 11/780 (a 1 MIP machine) that two 64-bit adders were equivalent in 95.8 minutes. He used an ordering that he derived manually.

Normalizing for technology, modern BDD packages are two orders of magnitude faster than Bryant's original implementation. A large source the improvement comes from the use of the *strong canonical form*, wherein a global database of BDD nodes is maintained, and no new node is added without checking to see if a node with the same label and 0- and 1-children exists in the database [3]. (For this approach to work, it is also required that the children of any node being added be in strong canonical form.) Other improvements stem from the use of complement pointers (if a pointer has its least-significant bit set, it refers to the complement of the function), better memory management (garbage collection based on reference counts, keeping nodes that are commonly accessed together close in memory), better hash functions, and better organization of the computed table (which keeps track of sub-problems that have already been encountered) [16].

Data Sets

The SIS (http://embedded.eecs.berkeley.edu/pubs/downloads/sis/) system from UC Berkeley is used for logic synthesis. It comes with a number of combinational and sequential circuits that have been used for benchmarking BDD packages.

The VIS (http://embedded.eecs.berkeley.edu/pubs/downloads/vis) system from UC Berkeley and UC Boulder is used for design verification; it uses BDDs to perform checks. The distribution includes a large collection of verification problems, ranging from simple hardware circuits, to complex multiprocessor cache systems.

URL to Code

A number of BDD packages exist today, but the package of choice is CUDD (http://vlsi.colorado.edu/~fabio/CUDD/). CUDD implements all the core features for manipulating BDDs, as well as variants. It is written in C++, and has extensive user and programmer documentation.

Cross References

► Symbolic Model Checking

Recommended Reading

1. Aziz, A., Tasiran, S., Brayton, R.: BDD Variable Ordering for Interacting Finite State Machines. In: ACM Design Automation Conference, pp. 283–288. (1994)
2. Berman, C.L.: Ordered Binary Decision Diagrams and Circuit Structure. In: IEEE International Conference on Computer Design. (1989)
3. Brace, K., Rudell, R., Bryant, R.: Efficient Implementation of a BDD Package. In: ACM Design Automation Conference. (1990)
4. Brayton, R., Hachtel, G., McMullen, C., Sangiovanni-Vincentelli, A.: Logic Minimization Algorithms for VLSI Synthesis. Kluwer Academic Publishers (1984)

5. Bryant, R.: Graph-based Algorithms for Boolean Function Manipulation. IEEE Transac. Comp. **C-35**, 677–691 (1986)
6. Cormen, T.H., Leiserson, C.E., Rivest, R.H., Stein, C.: Introduction to Algorithms. MIT Press (2001)
7. Garey, M.R., Johnson, D.S.: Computers and Intractability. W.H. Freeman and Co. (1979)
8. Gupta, A.: Formal Hardware Verification Methods: A Survey. Formal Method Syst. Des. **1**, 151–238 (1993)
9. Karchmer, M.: Communication Complexity: A New Approach to Circuit Depth. MIT Press (1989)
10. Kuehlmann, A., Krohm, F.: Equivalence Checking Using Cuts and Heaps. In: ACM Design Automation Conference (1997)
11. Malik, S., Wang, A.R., Brayton, R.K., Sangiovanni-Vincentelli, A.: Logic Verification using Binary Decision Diagrams in a Logic Synthesis Environment. In: IEEE International Conference on Computer-Aided Design, pp. 6–9. (1988)
12. McMillan, K.L.: Symbolic Model Checking. Kluwer Academic Publishers (1993)
13. De Micheli, G.: Synthesis and Optimization of Digital Circuits. McGraw Hill (1994)
14. Schoning, U., Pruim, R.: Gems of Theoretical Computer Science. Springer (1998)
15. Sipser, M.: Introduction to the Theory of Computation, 2nd edn. Course Technology (2005)
16. Somenzi, F.: Colorado University Decision Diagram Package. http://vlsi.colorado.edu/~fabio/
17. Wegener, I.: Branching Programs and Binary Decision Diagrams. SIAM (2000)
18. Yuan, J., Pixley, C., Aziz, A.: Constraint-Based Verfication. Springer (2006)

Bin Packing

1997; Coffman, Garay, Johnson

DAVID S. JOHNSON
Algorithms and Optimization Research Department, AT&T Labs, Florham Park, NJ, USA

Keywords and Synonyms

Cutting stock problem

Problem Definition

In the one-dimensional bin packing problem, one is given a list $L = (a_1, a_2, \ldots, a_n)$ of items, each item a_i having a size $s(a_i) \in (0, 1]$. The goal is to pack the items into a minimum number of unit-capacity bins, that is, to partition the items into a minimum number of sets, each having total size of at most 1. This problem is NP-hard, and so much of the research on it has concerned the design and analysis of approximation algorithms, which will be the subject of this article.

Although bin packing has many applications, it is perhaps most important for the role it has played as a proving ground for new algorithmic and analytical techniques. Some of the first worst- and average-case results for approximation algorithms were proved in this domain, as well as the first lower bounds on the competitive ratios of online algorithms. Readers interested in a more detailed coverage than is possible here are directed to two relatively recent surveys [4,11].

Key Results

Worst-Case Behavior

Asymptotic Worst-Case Ratios For most minimization problems, the standard worst-case metric for an approximation algorithm A is the maximum, over all instances I, of the ratio $A(I)/OPT(I)$, where $A(I)$ is the value of the solution generated by A and $OPT(I)$ is the optimal solution value. In the case of bin packing, however, there are limitations to this "absolute worst-case ratio" metric. Here it is already NP-hard to determine whether $OPT(I) = 2$, and hence no polynomial-time approximation algorithm can have an absolute worst-case ratio better than 1.5 unless P = NP. To better understand the behavior of bin packing algorithms in the typical situation where the given list L requires a large number of bins, researchers thus use a more refined metric for bin packing, the *asymptotic worst-case ratio* R_A^∞. This is defined in two steps as follows.

$$R_A^n = \max\{A(L)/OPT(L) : L \text{ is a list with } OPT(L) = n\}$$
$$R_A^\infty = \limsup_{n\to\infty} R_A^n$$

The first algorithm whose behavior was analyzed in these terms was *First Fit* (FF). This algorithm envisions an infinite sequence of empty bins $B_1, B_2, \ldots$ and, starting with the first item in the input list L, places each item in turn into the first bin which still has room for it. In a technical report from 1971 which was one of the very first papers in which worst-case performance ratios were studied, Ullman [22] proved the following.

Theorem 1 ([22]) $R_{FF}^\infty = 17/10$.

In addition to FF, five other simple heuristics received early study and have served as the inspiration for later research. *Best Fit* (BF) is the variant of FF in which each item is placed in the bin into which it will fit with the least space left over, with ties broken in favor of the earliest such bin. Both FF and BF can be implemented to run in time $O(n \log n)$ [12]. *Next Fit* (NF) is a still simpler and linear-time algorithm in which the first item is placed in the first bin, and thereafter each item is placed in the last nonempty bin if it will fit, otherwise a new bin is started. *First Fit Decreasing* (FFD) and *Best Fit Decreasing* (BFD) are the variants of those algorithms in which the input list is first sorted into nonincreasing order by size and then the corresponding packing rule is applied. The results for these algorithms are as follows.

Theorem 2 ([12]) $R^{\infty}_{NF} = 2$.

Theorem 3 ([13]) $R^{\infty}_{BF} = 17/10$.

Theorem 4 ([12,13]) $R^{\infty}_{FFD} = R^{\infty}_{BFD} = 11/9 = 1.222\ldots$

The abovementioned algorithms are relatively simple and intuitive. If one is willing to consider more complicated algorithms, one can do substantially better. The current best polynomial-time bin packing algorithm is very good indeed. This is the 1982 algorithm of Karmarkar and Karp [15], denoted here as "KK." It exploits the ellipsoid algorithm, approximation algorithms for the knapsack problem, and a clever rounding scheme to obtain the following guarantees.

Theorem 5 ([15]) $R^{\infty}_{KK} = 1$ *and there is a constant* c *such that for all lists* L,

$$KK(L) \le OPT(L) + c \log^2(OPT(L)) .$$

Unfortunately, the running time for KK appears to be worse than $O(n^8)$, and BFD and FFD remain much more practical alternatives.

Online Algorithms Three of the abovementioned algorithms (FF, BF, and NF) are *online* algorithms, in that they pack items in the order given, without reference to the sizes or number of later items. As was subsequently observed in many contexts, the online restriction can seriously limit the ability of an algorithm to produce good solutions. Perhaps the first limitation of this type to be proved was Yao's theorem [24] that no online algorithm A for bin packing can have $R^{\infty}_{A} < 1.5$. The bound has since been improved to the following.

Theorem 6 ([23]) *If* A *is an online algorithm for bin packing, then* $R^{\infty}_{A} \ge 1.540\ldots$

Here the exact value of the lower bound is the solution to a complicated linear program.

Yao's paper also presented an online algorithm *Revised First Fit* (RFF) that had $R^{\infty}_{RFF} = 5/3 = 1.666\ldots$ and hence got closer to this lower bound than FF and BF. This algorithm worked by dividing the items into four classes based on size and index, and then using different packing rules (and packings) for each class. Subsequent algorithms improved on this by going to more and more classes. The current champion is the online *Harmonic++* algorithm (H++) of [21]:

Theorem 7 ([21]) $R^{\infty}_{H++} \le 1.58889$.

Bounded-Space Algorithms The NF algorithm, in addition to being online, has another property worth noting: no more than a constant number of partially filled bins remain open to receive additional items at any given time. In the case of NF, the constant is 1 – only the last partially filled bin can receive additional items. Bounding the number of open bins may be necessary in some applications, such as packing trucks on loading docks. The bounded-space constraint imposes additional limits on algorithmic behavior however.

Theorem 8 ([17]) *For any online bounded-space algorithm* A, $R^{\infty}_{A} \ge 1.691\ldots$.

The constant $1.691\ldots$ arises in many other bin packing contexts. It is commonly denoted by h_∞ and equals $\sum_{i=1}^{\infty}(1/t_i)$, where $t_1 = 1$ and, for $i > 1$, $t_i = t_{i-1}(t_{i-1}+1)$.

The lower bound in Theorem 8 is tight, owing to the existence of the *Harmonic*$_k$ algorithms (H$_k$) of [17]. H$_k$ is a class-based algorithm in which the items are divided into classes C_h, $1 \le h \le k$, with C_k consisting of all items with size $1/k$ or smaller, and C_h, $1 \le h < k$, consisting of all a_i with $1/(h+1) < s(a_i) \le 1/h$. The items in each class are then packed by NF into a separate packing devoted just to that class. Thus, at most k bins are open at any time. In [17] it was shown that $\lim_{k\to\infty} R^{\infty}_{H_k} = h_\infty = 1.691\ldots$. This is even better than the asymptotic worst-case ratio of 1.7 for the unbounded-space algorithms FF and BF, although it should be noted that the bounded-space variant of BF in which all but the two most-full bins are closed also has $R^{\infty}_{A} = 1.7$ [8].

Average-Case Behavior

Continuous Distributions Bin packing also served as an early test bed for studying the average-case behavior of approximation algorithms. Suppose F is a distribution on $(0, 1]$ and L_n is a list of n items with item sizes chosen independently according to F. For any list L, let $s(L)$ denote the lower bound on $OPT(L)$ obtained by summing the sizes of all the items in L. Then define

$$ER^n_A(F) = E\left[A(L_n)/OPT(L_n)\right] ,$$
$$ER^{\infty}_A(F) = \limsup_{n\to\infty} ER^n_A$$
$$EW^n_A(F) = E\left[A(L_n) - s(L_n)\right]$$

The last definition is included since $ER^{\infty}_A(F) = 1$ occurs frequently enough that finer distinctions are meaningful. For example, in the early 1980s, it was observed that for the distribution $F = U(0, 1]$ in which item sizes are uniformly distributed on the interval $(0, 1]$, $ER^{\infty}_{FFD}(F) = ER^{\infty}_{BFD}(F) = 1$, as a consequence of the following more-detailed results.

Theorem 9 ([16,20]) *For $A \in \{FFD, BFD, OPT\}$, $EW_A^n(U(0,1]) = \Theta(\sqrt{n})$.*

Somewhat surprisingly, it was later discovered that the online FF and BF algorithms also had sublinear expected waste, and hence $ER_A^\infty(U(0,1]) = 1$.

Theorem 10 ([5,19])

$$EW_{FF}^n(U(0,1]) = \Theta(n^{2/3})$$
$$EW_{BF}^n(U(0,1]) = \Theta(n^{1/2}\log^{3/4} n)$$

This good behavior does not, however, extend to the bounded-space algorithms NF and H$_k$:

Theorem 11 ([6,18])

$$ER_{NF}^\infty(U(0,1]) = 4/3 = 1.333\ldots$$
$$\lim_{k\to\infty} ER_{H_k}(U(0,1]) = \pi^2/3 - 2 = 1.2899\ldots$$

All the above results except the last two exploit the fact that the distribution $U(0,1]$ is symmetric about 1/2, and hence an optimal packing consists primarily of two-item bins, with items of size $s > 1/2$ matched with smaller items of size very close to $1-s$. The proofs essentially show that the algorithms in question do good jobs of constructing such matchings. In practice, however, there will clearly be situations where more than matching is required. To model such situations, researchers first turned to the distributions $U(0,b]$, $0 < b < 1$, where item sizes are chosen uniformly from the interval $(0,b]$. Simulations suggest that such distributions make things worse for the online algorithms FF and BF, which appear to have $ER_A^\infty(U(0,b]) > 1$ for all $b \in (0,1)$. Surprisingly, they make things better for FFD and BFD (and the optimal packing).

Theorem 12 ([2,14])
1. *For $0 < b \le 1/2$ and $A \in \{FFD, BFD\}$, $EW_A^n(U(0,b]) = O(1)$.*
2. *For $1/2 < b < 1$ and $A \in \{FFD, BFD\}$, $EW_A^n(U(0,b]) = \Theta(n^{1/3})$.*
3. *For $0 < b < 1$, $EW_{OPT}^n(U(0,b]) = O(1)$.*

Discrete Distributions In many applications, the item sizes come from a finite set, rather than a continuous distribution like those discussed above. Thus, recently the study of average-case behavior for bin packing has turned to *discrete distributions*. Such a distribution is specified by a finite list $s_1, s_2, \ldots, s_d$ of rational sizes and for each s_i a corresponding rational probability p_i. A remarkable result of Courcoubetis and Weber [7] says the following.

Theorem 13 ([7]) *For any discrete distribution F, $EW_{OPT}^n(F)$ is either $\Theta(n)$, $\Theta(\sqrt{n})$, or O(1).*

The discrete analogue of the continuous distribution $U(0,b]$ is the distribution $U\{j,k\}$, where the sizes are $1/k, 2/k, \ldots, j/k$ and all the probabilities equal $1/j$. Simulations suggest that the behavior of FF and BF in the discrete case are qualitatively similar to the behavior in the continuous case, whereas the behavior of FFD and BFD is considerably more bizarre [3]. Of particular note is the distribution $F = U\{6,13\}$, for which $ER_{FFD}^\infty(F)$ is strictly greater than $ER_{FF}^\infty(F)$, in contrast to all the previously implied comparisons between the two algorithms.

For discrete distributions, however, the standard algorithms are all dominated by a new online algorithm called the *Sum-of-Squares* (SS) algorithm. Note that since the item sizes are all rational, they can be scaled so that they (and the bin size B) are all integral. Then at any given point in the operation of an online algorithm, the current packing can be summarized by giving, for each h, $1 \le h \le B$, the number n_h of bins containing items of total size h. In SS, one packs each item so as to minimize $\sum_{h=1}^{B-1} n_h^2$.

Theorem 14 ([9]) *For any discrete distribution F, the following hold.*
1. *If $EW_{OPT}^n(F) = \Theta(\sqrt{n})$, then $EW_{SS}^n(F) = \Theta(\sqrt{n})$.*
2. *If $EW_{OPT}^n(F) = O(1)$, then $EW_{SS}^n(F) \in \{O(1), \Theta(\log n)\}$.*

In addition, a simple modification to SS can eliminate the $\Theta(\log n)$ case of condition 2.

Applications

There are many potential applications of one-dimensional bin packing, from packing bandwidth requests into fixed-capacity channels to packing commercials into station breaks. In practice, simple heuristics like FFD and BFD are commonly used.

Open Problems

Perhaps the most fundamental open problem related to bin packing is the following. As observed above, there is a polynomial-time algorithm (KK) whose packings are within $O(\log^2(OPT))$ bins of optimal. Is it possible to do better? As far as is currently known, there could still be a polynomial-time algorithm that always gets within one bin of optimal, even if P $\neq$ NP.

Experimental Results

Bin packing has been a fertile ground for experimental analysis, and many of the theorems mentioned above were first conjectured on the basis of experimental results. For example, the experiments reported in [1] inspired Theorems 10 and 12, and the experiments in [10] inspired Theorem 14.

Cross References

► Approximation Schemes for Bin Packing

Recommended Reading

1. Bentley, J.L., Johnson, D.S., Leighton, F.T., McGeoch, C.C.: An experimental study of bin packing. In: Proc. of the 21st Annual Allerton Conference on Communication, Control, and Computing, Urbana, University of Illinois, 1983 pp. 51–60
2. Bentley, J.L., Johnson, D.S., Leighton, F.T., McGeoch, C.C., McGeoch, L.A.: Some unexpected expected behavior results for bin packing. In: Proc. of the 16th Annual ACM Symposium on Theory of Computing, pp. 279–288. ACM, New York (1984)
3. Coffman Jr, E.G., Courcoubetis, C., Garey, M.R., Johnson, D.S., McGeoch, L.A., Shor, P.W., Weber, R.R., Yannakakis, M.: Fundamental discrepancies between average-case analyses under discrete and continuous distributions. In: Proc. of the 23rd Annual ACM Symposium on Theory of Computing, New York, 1991, pp. 230–240. ACM Press, New York (1991)
4. Coffman Jr., E.G., Garey, M.R., Johnson, D.S.: Approximation algorithms for bin-packing: A survey. In: Hochbaum, D. (ed.) Approximation Algorithms for NP-Hard Problems, pp. 46–93. PWS Publishing, Boston (1997)
5. Coffman Jr., E.G., Johnson, D.S., Shor, P.W., Weber, R.R.: Bin packing with discrete item sizes, part II: Tight bounds on first fit. Random Struct. Algorithms **10**, 69–101 (1997)
6. Coffman Jr., E.G., So, K., Hofri, M., Yao, A.C.: A stochastic model of bin-packing. Inf. Cont. **44**, 105–115 (1980)
7. Courcoubetis, C., Weber, R.R.: Necessary and sufficient conditions for stability of a bin packing system. J. Appl. Prob. **23**, 989–999 (1986)
8. Csirik, J., Johnson, D.S.: Bounded space on-line bin packing: Best is better than first. Algorithmica **31**, 115–138 (2001)
9. Csirik, J., Johnson, D.S., Kenyon, C., Orlin, J.B., Shor, P.W., Weber, R.R.: On the sum-of-squares algorithm for bin packing. J. ACM **53**, 1–65 (2006)
10. Csirik, J., Johnson, D.S., Kenyon, C., Shor, P.W., Weber, R.R.: A self organizing bin packing heuristic. In: Proc. of the 1999 Workshop on Algorithm Engineering and Experimentation. LNCS, vol. 1619, pp. 246–265. Springer, Berlin (1999)
11. Galambos, G., Woeginger, G.J.: Online bin packing – a restricted survey. ZOR Math. Methods Oper. Res. **42**, 25–45 (1995)
12. Johnson, D.S.: Near-Optimal Bin Packing Algorithms. Ph. D. thesis, Massachusetts Institute of Technology, Department of Mathematics, Cambridge (1973)
13. Johnson, D.S., Demers, A., Ullman, J.D., Garey, M.R., Graham, R.L.: Worst-case performance bounds for simple one-dimensional packing algorithms. SIAM J. Comput. **3**, 299–325 (1974)
14. Johnson, D.S., Leighton, F.T., Shor, P.W., Weber, R.R.: The expected behavior of FFD, BFD, and optimal bin packing under $U(0, \alpha])$ distributions (in preparation)
15. Karmarkar, N., Karp, R.M.: An efficient approximation scheme for the one-dimensional bin packing problem. In: Proc. of the 23rd Annual Symposium on Foundations of Computer Science, pp. 312–320. IEEE Computer Soc, Los Alamitos, CA (1982)
16. Knödel, W.: A bin packing algorithm with complexity $O(n \log n)$ in the stochastic limit. In: Proc. 10th Symp. on Mathematical Foundations of Computer Science. LNCS, vol. 118, pp. 369–378. Springer, Berlin (1981)
17. Lee, C.C., Lee, D.T.: A simple on-line packing algorithm. J. ACM **32**, 562–572 (1985)
18. Lee, C.C., Lee, D.T.: Robust on-line bin packing algorithms. Tech. Rep. Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL (1987)
19. Leighton, T., Shor, P.: Tight bounds for minimax grid matching with applications to the average case analysis of algorithms. Combinatorica **9** 161–187 (1989)
20. Lueker, G.S.: An average-case analysis of bin packing with uniformly distributed item sizes. Tech. Rep. Report No 181, Dept. of Information and Computer Science, University of California, Irvine, CA (1982)
21. Seiden, S.S.: On the online bin packing problem. J. ACM **49**, 640–671 (2002)
22. Ullman, J.D.: The performance of a memory allocation algorithm. Tech. Rep. 100, Princeton University, Princeton, NJ (1971)
23. van Vliet, A.: An improved lower bound for on-line bin packing algorithms. Inf. Proc. Lett. **43**, 277–284 (1992)
24. Yao, A.C.: New algorithms for bin packing. J. ACM **27**, 207–227 (1980)

Block Edit Distance

► Edit Distance Under Block Operations

Block-Sorting Data Compression

► Burrows–Wheeler Transform

Boolean Formulas

► Learning Automata

Boolean Satisfiability

► Exact Algorithms for General CNF SAT

Boosting Textual Compression

2005; Ferragina, Giancarlo, Manzini, Sciortino

PAOLO FERRAGINA[1], GIOVANNI MANZINI[2]

[1] Department of Computer Science, University of Pisa, Pisa, Italy

[2] Department of Computer Science, University of Eastern Piedmont, Alessandria, Italy

Keywords and Synonyms

High-order compression models; Context-aware compression

Problem Definition

Informally, a boosting technique is a method that, when applied to a particular class of algorithms, yields improved algorithms. The improvement must be provable and well defined in terms of one or more of the parameters characterizing the algorithmic performance. Examples of boosters can be found in the context of Randomized Algorithms (here, a booster allows one to turn a BPP algorithm into an RP one [6]) and Computational Learning Theory (here, a booster allows one to improve the prediction accuracy of a weak learning algorithm [10]). The problem of Compression Boosting consists of designing a technique that improves the compression performance of a wide class of algorithms. In particular, the results of Ferragina et al. provide a general technique for turning a compressor that uses no context information into one that always uses the best possible context.

The classic Huffman and Arithmetic coding algorithms [1] are examples of *statistical* compressors which typically encode an input symbol according to its *overall* frequency in the data to be compressed.[1] This approach is efficient and easy to implement but achieves poor compression. The compression performance of statistical compressors can be improved by adopting *higher*-order models that obtain better estimates for the frequencies of the input symbols. The PPM compressor [9] implements this idea by collecting (the frequency of) all symbols which follow *any* k-long context, and by compressing them via Arithmetic coding. The length k of the context is a parameter of the algorithm that depends on the data to be compressed: it is different if one is compressing English text, a DNA sequence, or an XML document. There exist other examples of sophisticated compressors that use context information in an *implicit* way, such as Lempel–Ziv and Burrows–Wheeler compressors [9]. All these context-aware algorithms are effective in terms of compression performance, but are usually rather complex to implement and difficult to analyze.

Applying the boosting technique of Ferragina et al. to Huffman or Arithmetic Coding yields a new compression algorithm with the following features: (*i*) the new algorithm uses the boosted compressor as a black box, (*ii*) the new algorithm compresses in a PPM-like style, automatically choosing the *optimal* value of k, (*iii*) the new algorithm has essentially the same time/space asymptotic performance of the boosted compressor. The following sections give a precise and formal treatment of the three properties (*i*)–(*iii*) outlined above.

[1]In their dynamic versions these algorithms consider the frequency of a symbol in the already scanned portion of the input.

Key Results

Notation: The Empirical Entropy

Let s be a string over the alphabet $\Sigma = \{a_1, \ldots, a_h\}$ and, for each $a_i \in \Sigma$, let n_i be the number of occurrences of a_i in s. The *0th order empirical entropy* of the string s is defined as $H_0(s) = -\sum_{i=1}^{h}(n_i/|s|)\log(n_i/|s|)$, where it is assumed that all logarithms are taken to the base 2 and $0\log 0 = 0$. It is well known that H_0 is the maximum compression one can achieve using a uniquely decodable code in which a fixed codeword is assigned to each alphabet symbol. Greater compression is achievable if the codeword of a symbol depends on the k symbols following it (namely, its *context*).[2] Let us define w_s as the string of single symbols immediately preceding the occurrences of w in s. For example, for $s = \texttt{bcabcabdca}$ it is $\texttt{ca}_s = \texttt{bbd}$. The value

$$H_k(s) = \frac{1}{|s|}\sum_{w\in\Sigma^k} |w_s|\, H_0(w_s) \tag{1}$$

is the k-th order empirical entropy of s and is a lower bound to the compression one can achieve using codewords which only depend on the k symbols immediately following the one to be encoded.

Example 1 Let $s = \texttt{mississippi}$. For $k = 1$ it is $\texttt{i}_s = \texttt{mssp}$, $\texttt{s}_s = \texttt{isis}$, $\texttt{p}_s = \texttt{ip}$. Hence,

$$\begin{aligned} H_1(s) &= \frac{4}{11}H_0(\texttt{mssp}) + \frac{4}{11}H_0(\texttt{isis}) + \frac{2}{11}H_0(\texttt{ip}) \\ &= \frac{6}{11} + \frac{4}{11} + \frac{2}{11} = \frac{12}{11}\,. \end{aligned}$$

Note that the empirical entropy is defined for any string and can be used to measure the performance of compression algorithms without any assumption on the input source. Unfortunately, for some (highly compressible) strings, the empirical entropy provides a lower bound that is too conservative. For example, for $s = \texttt{a}^n$ it is $|s|\, H_k(s) = 0$ for any $k \geq 0$. To better deal with highly

[2]In data compression it is customary to define the context looking at the symbols *preceding* the one to be encoded. The present entry uses the non-standard "forward" contexts to simplify the notation of the following sections. Note that working with "forward" contexts is equivalent to working with the traditional "backward" contexts on the string s reversed (see [3] for details).

compressible strings [7] introduced the notion of 0*th order modified empirical* entropy $H_0^*(s)$ whose property is that $|s|H_0^*(s)$ is at least equal to the number of bits needed to write down the length of s in binary. The *kth order modified empirical entropy* H_k^* is then defined in terms of H_0^* as the maximum compression one can achieve by looking at *no more than* k symbols following the one to be encoded.

The Burrows–Wheeler Transform

Given a string s, the Burrows–Wheeler transform [2] (bwt) consists of three basic steps: (1) append to the end of s a special symbol \$ smaller than any other symbol in Σ; (2) form a *conceptual* matrix $\mathcal{M}$ whose rows are the cyclic shifts of the string $s\$$, sorted in lexicographic order; (3) construct the transformed text $\hat{s} = \mathtt{bwt}(s)$ by taking the last column of $\mathcal{M}$ (see Fig. 1). In [2] Burrows and Wheeler proved that $\hat{s}$ is a permutation of s, and that from $\hat{s}$ it is possible to recover s in $O(|s|)$ time.

To see the power of the bwt the reader should reason in terms of empirical entropy. Fix a positive integer k. The first k columns of the bwt matrix contain, lexicographically ordered, all length-k substrings of s (and k substrings containing the symbol \$). For any length-k substring w of s, the symbols immediately preceding every occurrence of w in s are grouped together in a set of consecutive positions of $\hat{s}$ since they are the last symbols of the rows of $\mathcal{M}$ prefixed by w. Using the notation introduced for defining H_k, it is possible to rephrase this property by saying that the symbols of w_s are consecutive within $\hat{s}$, or equivalently, that $\hat{s}$ contains, as a substring, a permutation $\pi_w(w_s)$ of the string w_s.

Example 2 Let $s = \mathtt{mississippi}$ and $k = 1$. Figure 1 shows that $\hat{s}[1,4] = \mathtt{pssm}$ is a permutation of $\mathtt{i}_s = \mathtt{mssp}$. In addition, $\hat{s}[6,7] = \mathtt{pi}$ is a permutation of $\mathtt{p}_s = \mathtt{ip}$, and $\hat{s}[8,11] = \mathtt{ssii}$ is a permutation of $\mathtt{s}_s = \mathtt{isis}$.

Since permuting a string does not change its (modified) 0th order empirical entropy (that is, $H_0(\pi_w(w_s)) = H_0(w_s)$), the Burrows–Wheeler transform can be seen as a tool for reducing the problem of compressing s up to its kth order entropy to the problem of compressing *distinct portions* of $\hat{s}$ up to their 0*th order* entropy. To see this, assume partitioning of $\hat{s}$ into the substrings $\pi_w(w_s)$ by varying w over Σ^k. It follows that $\hat{s} = \bigsqcup_{w\in\Sigma^k} \pi_w(w_s)$ where $\bigsqcup$ denotes the concatenation operator among strings.[3]

[3] In addition to $\sqcup_{w\in\Sigma^k} \pi_w(w_s)$, the string $\hat{s}$ also contains the last k symbols of s (which do not belong to any w_s) and the special symbol \$. For simplicity these symbols will be ignored in the following part of the entry.

By (1) it follows that

$$\sum_{w\in\Sigma^k} |\pi_w(w_s)| H_0(\pi_w(w_s)) = \sum_{w\in\Sigma^k} |w_s| H_0(w_s) = |s| H_k(s).$$

Hence, to compress s up to $|s|\, H_k(s)$ it suffices to compress each substring $\pi_w(w_s)$ up to its 0th order empirical entropy. Note, however, that in the above scheme the parameter k must be chosen in advance. Moreover, a similar scheme cannot be applied to H_k^* which is defined in terms of contexts of length *at most* k. As a result, no efficient procedure is known for computing the partition of $\hat{s}$ corresponding to $H_k^*(s)$. The compression booster [3] is a natural complement to the bwt and allows one to compress any string s up to $H_k(s)$ (or $H_k^*(s)$) simultaneously for all $k \geq 0$.

The Compression Boosting Algorithm

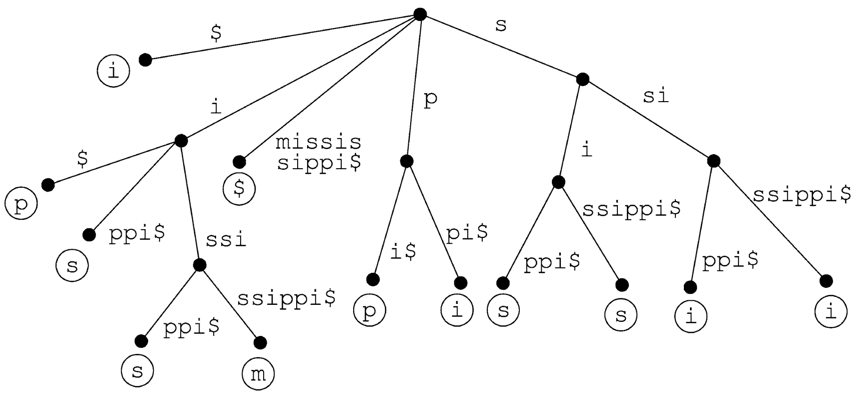
A crucial ingredient of compression boosting is the relationship between the bwt matrix and the suffix tree data structure. Let $\mathcal{T}$ denote the suffix tree of the string $s\$$. $\mathcal{T}$ has $|s| + 1$ leaves, one per suffix of $s\$$, and edges labeled with substrings of $s\$$ (see Fig. 1). Any node u of $\mathcal{T}$ has *implicitly associated* a substring of $s\$$, given by the concatenation of the edge labels on the downward path from the root of $\mathcal{T}$ to u. In that implicit association, the leaves of $\mathcal{T}$ correspond to the suffixes of $s\$$. Assume that the suffix tree edges are sorted lexicographically. Since each row of the bwt matrix is prefixed by one suffix of $s\$$ and rows are lexicographically sorted, the ith leaf (counting from the left) of the suffix tree corresponds to the ith row of the bwt matrix. Associate to the ith leaf of $\mathcal{T}$ the ith symbol of $\hat{s} = \mathtt{bwt}(s)$. In Fig. 1 these symbols are represented inside circles.

For any suffix tree node u, let $\hat{s}\langle u\rangle$ denote the substring of $\hat{s}$ obtained by concatenating, from left to right, the symbols associated to the leaves descending from node u. Of course $\hat{s}\langle root(\mathcal{T})\rangle = \hat{s}$. A subset $\mathcal{L}$ of $\mathcal{T}$'s nodes is called a *leaf cover* if every leaf of the suffix tree has a *unique* ancestor in $\mathcal{L}$. Any leaf cover $\mathcal{L} = \{u_1, \ldots, u_p\}$ naturally induces a partition of the leaves of $\mathcal{T}$. Because of the relationship between $\mathcal{T}$ and the bwt matrix this is also a partition of $\hat{s}$, namely $\{\hat{s}\langle u_1\rangle, \ldots, \hat{s}\langle u_p\rangle\}$.

Example 3 Consider the suffix tree in Fig. 1. A leaf cover consists of all nodes of depth one. The partition of $\hat{s}$ induced by this leaf cover is $\{\mathtt{i}, \mathtt{pssm}, \$, \mathtt{pi}, \mathtt{ssii}\}$.

Let C denote a function that associates to every string x over $\Sigma \cup \{\$\}$ a positive real value $C(x)$. For any leaf cover $\mathcal{L}$, define its cost as $C(\mathcal{L}) = \sum_{u\in\mathcal{L}} C(\hat{s}\langle u\rangle)$. In other words, the cost of the leaf cover $\mathcal{L}$ is equal to the sum of the costs of the strings in the partition induced by $\mathcal{L}$. A leaf cover

```
$ mississipp i
i $mississip p
i ppi$missis s
i ssippi$mis s
i ssissippi$ m
m ississippi $
p i$mississi p
p pi$mississ i
s ippi$missi s
s issippi$mi s
s sippi$miss i
s sissippi$m i
```



Boosting Textual Compression, Figure 1
The bwt matrix (*left*) and the suffix tree (*right*) for the string s = mississippi$. The output of the bwt is the last column of the bwt matrix, i. e., $\hat{s}$ = bwt(s) = ipssm$pissii

$\mathcal{L}_{\min}$ is called *optimal* with respect to C if $C(\mathcal{L}_{\min}) \leq C(\mathcal{L})$, for any leaf cover $\mathcal{L}$.

Let $\mathbb{A}$ be a compressor such that, for any string x, its output size is bounded by $|x|H_0(x) + \eta|x| + \mu$ bits, where η and μ are constants. Define the cost function $C_{\mathbb{A}}(x) = |x|H_0(x) + \eta|x| + \mu$. In [3] Ferragina et al. exhibit a linear-time greedy algorithm that computes the optimal leaf cover $\mathcal{L}_{\min}$ with respect to $C_{\mathbb{A}}$. The authors of [3] also show that, for any $k \geq 0$, there exists a leaf cover $\mathcal{L}_k$ of cost $C_{\mathbb{A}}(\mathcal{L}_k) = |s|\ H_k(s) + \eta|s| + O(|\Sigma|^k)$. These two crucial observations show that, if one uses $\mathbb{A}$ to compress each substring in the partition induced by the optimal leaf cover $\mathcal{L}_{\min}$, the total output size is bounded in terms of $|s|\ H_k(s)$, for any $k \geq 0$. In fact,

$$\sum_{u \in \mathcal{L}_{\min}} C_{\mathbb{A}}(\hat{s}\langle u \rangle) = C_{\mathbb{A}}(\mathcal{L}_{\min}) \leq C_{\mathbb{A}}(\mathcal{L}_k) = |s|\ H_k(s) + \eta|s| + O(|\Sigma|^k)$$

In summary, boosting the compressor $\mathbb{A}$ over the string s consists of three main steps:

1. Compute $\hat{s}$ = bwt(s);
2. Compute the optimal leaf cover $\mathcal{L}_{\min}$ with respect to $C_{\mathbb{A}}$, and partition $\hat{s}$ according to $\mathcal{L}_{\min}$;
3. Compress each substring of the partition using the algorithm $\mathbb{A}$.

So the boosting paradigm reduces the design of effective compressors that use context information, to the (usually easier) design of 0th order compressors. The performance of this paradigm is summarized by the following theorem.

Theorem 1 (Ferragina et al. 2005) *Let $\mathbb{A}$ be a compressor that squeezes any string x in at most $|x|H_0(x)+\eta|x|+\mu$ bits. The compression booster applied to $\mathbb{A}$ produces an output whose size is bounded by $|s|\ H_k(s) + \log|s| + \eta|s| + O(|\Sigma|^k)$ bits simultaneously for all $k \geq 0$. With respect to $\mathbb{A}$, the booster introduces a space overhead of $O(|s|\log|s|)$ bits and no asymptotic time overhead in the compression process.* □

A similar result holds for the modified entropy H_k^* as well (but it is much harder to prove): Given a compressor $\mathbb{A}$ that squeezes any string x in at most $\lambda|x|\ H_0^*(x) + \mu$ bits, the compression booster produces an output whose size is bounded by $\lambda|s|\ H_k^*(s) + \log|s| + O(|\Sigma|^k)$ bits, simultaneously for all $k \geq 0$. In [3] the authors also show that no compression algorithm, satisfying some mild assumptions on its inner working, can achieve a similar bound in which both the multiplicative factor λ and the additive logarithmic term are dropped simultaneously. Furthermore [3] proposes an instantiation of the booster which compresses any string s in at most $2.5|s|H_k^*(s) + \log|s| + O(|\Sigma|^k)$ bits. This bound is analytically superior to the bounds proven for the best existing compressors including Lempel–Ziv, Burrows–Wheeler, and PPM compressors.

Applications

Apart from the natural application in data compression, compressor boosting has been used also to design Compressed Full-text Indexes [8].

Open Problems

The boosting paradigm may be generalized as follows: Given a compressor $\mathbb{A}$, find a permutation $\mathcal{P}$ for the symbols of the string s *and* a partitioning strategy such that the boosting approach, applied to them, minimizes the output size. These pages have provided convincing evidence that the Burrows–Wheeler Transform is an elegant and efficient permutation $\mathcal{P}$. Surprisingly enough, other classic Data Compression problems fall into this framework: Shortest Common Superstring (which is MAX-SNP hard), Run Length Encoding for a Set of Strings (which is polynomially solvable), LZ77 and minimum number of phrases (which is MAX-SNP-Hard). Therefore, the boosting approach is general enough to deserve further theoretical and practical attention [5].

Experimental Results

An investigation of several compression algorithms based on boosting, and a comparison with other state-of-the-art compressors is presented in [4]. The experiments show that the boosting technique is more robust than other bwt-based approaches, and works well even with less effective 0th order compressors. However, these positive features are achieved using more (time and space) resources.

Data Sets

The data sets used in [4] are available from http://www.mfn.unipmn.it/~manzini/boosting. Other data sets for compression and indexing are available at the Pizza&Chili site http://pizzachili.di.unipi.it/.

URL to Code

The Compression Boosting page (http://www.mfn.unipmn.it/~manzini/boosting) contains the source code of all the algorithms tested in [4]. The code is organized in a highly modular library that can be used to boost any compressor even without knowing the bwt or the boosting procedure.

Cross References

- ▶ Arithmetic Coding for Data Compression
- ▶ Burrows–Wheeler Transform
- ▶ Compressed Text Indexing
- ▶ Table Compression
- ▶ Tree Compression and Indexing

Recommended Reading

1. Bell, T.C., Cleary, J.G., Witten, I.H.: Text compression. Prentice Hall, NJ (1990)
2. Burrows, M. Wheeler, D.: A block sorting lossless data compression algorithm. Tech. Report 124, Digital Equipment Corporation (1994)
3. Ferragina, P., Giancarlo, R., Manzini, G., Sciortino, M.: Boosting textual compression in optimal linear time. J. ACM **52**, 688–713 (2005)
4. Ferragina, P., Giancarlo, R., Manzini, G.: The engineering of a compression boosting library: Theory vs practice in bwt compression. In: Proc. 14th European Symposium on Algorithms (ESA). LNCS, vol. 4168, pp. 756–767. Springer, Berlin (2006)
5. Giancarlo, R., Restivo, A., Sciortino, M.: From first principles to the Burrows and Wheeler transform and beyond, via combinatorial optimization. Theor. Comput. Sci. **387**(3):236-248 (2007)
6. Karp, R., Pippenger, N., Sipser, M.: A Time-Randomness tradeoff. In: Proc. Conference on Probabilistic Computational Complexity, AMS, 1985, pp. 150–159
7. Manzini, G.: An analysis of the Burrows-Wheeler transform. J. ACM **48**, 407–430 (2001)
8. Navarro, G., Mäkinen, V.: Compressed full text indexes. ACM Comput. Surv. **39**(1) (2007)
9. Salomon, D.: Data Compression: the Complete Reference, 4th edn. Springer, London (2004)
10. Schapire, R.E.: The strength of weak learnability. Mach. Learn. **2**, 197–227 (1990)

Branchwidth of Graphs

2003; Fomin, Thilikos

FEDOR FOMIN[1], DIMITRIOS THILIKOS[2]
[1] Department of Informatics, University of Bergen, Bergen, Norway
[2] Department of Mathematics, National and Kapodistrian University of Athens, Athens, Greece

Keywords and Synonyms

Tangle Number

Problem Definition

Branchwidth, along with its better-known counterpart, treewidth, are measures of the “global connectivity” of a graph.

Definition

Let G be a graph on n vertices. A *branch decomposition* of G is a pair (T, τ), where T is a tree with vertices of degree 1 or 3 and τ is a bijection from the set of leaves of T to the edges of G. The *order*, we denote it as $\alpha(e)$, of an edge e in T is the number of vertices v of G such that there are leaves t_1, t_2 in T in different components of $T(V(T), E(T) - e)$ with $\tau(t_1)$ and $\tau(t_2)$ both containing v as an endpoint.

The *width* of (T, τ) is equal to $\max_{e \in E(T)}\{\alpha(e)\}$, i. e. is the maximum order over all edges of T. The *branchwidth* of G is the minimum width over all the branch decompositions of G (in the case where $|E(G)| \leq 1$, then we define the branchwidth to be 0; if $|E(G)| = 0$, then G has no branch decomposition; if $|E(G)| = 1$, then G has a branch decomposition consisting of a tree with one vertex – the width of this branch decomposition is considered to be 0).

The above definition can be directly extended to hypergraphs where τ is a bijection from the leaves of T to the hyperedges of G. The same definition can easily be extended to matroids.

Branchwidth was first defined by Robertson and Seymour in [25] and served as a main tool for their proof of Wagner's Conjecture in their Graph Minors series of papers. There, branchwidth was used as an alternative to the parameter of treewidth as it appeared easier to handle for the purposes of the proof. The relation between branchwidth and treewidth is given by the following result.

Theorem 1 **([25])** *If G is a graph, then* $\text{branchwidth}(G) \leq \text{treewidth}(G) + 1 \leq \lfloor 3/2\, \text{branchwidth}(G) \rfloor$.

The algorithmic problems related to branchwidth are of two kinds: first find fast algorithms computing its value and, second, use it in order to design fast dynamic programming algorithms for other problems.

Key Results

Algorithms for Branchwidth

Computing branchwidth is an NP-hard problem ([29]). Moreover, the problem remains NP-hard even if we restrict its input graphs to the class of split graphs or bipartite graphs [20].

On the positive side, branchwidth is computable in polynomial time on interval graphs [20,24], and circular arc graphs [21]. Perhaps the most celebrated positive result on branchwidth is an $O(n^2)$ algorithm for the branchwidth of planar graphs, given by Seymour and Thomas in [29]. In the same paper they also give an $O(n^4)$ algorithm to compute an optimal branch decomposition. (The running time of this algorithm has been improved to $O(n^3)$ in [18].) The algorithm in [29] is basically an algorithm for a parameter called carving width, related to telephone routing and the result for branchwidth follows from the fact that the branch width of a planar graph is half of the carving-width of its medial graph.

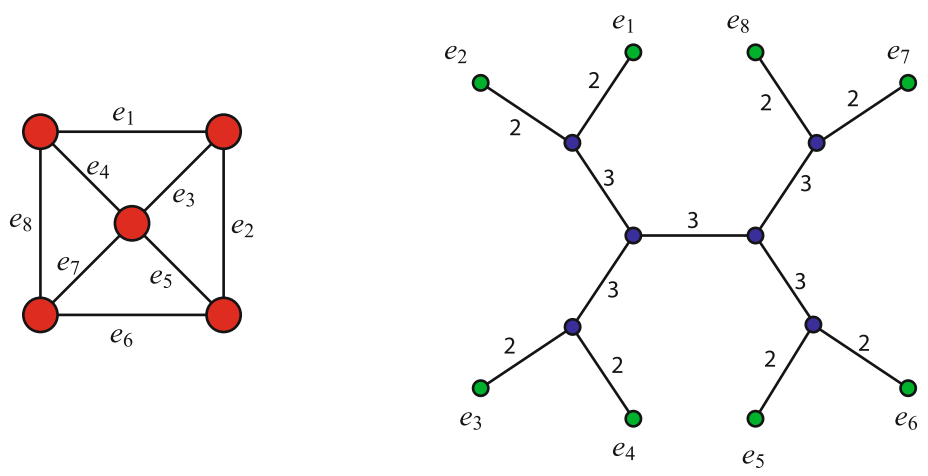
The algorithm for planar graphs [29] can be used to construct an approximation algorithm for branchwidth of some non-planar graphs. On graph classes excluding a single crossing graph as a minor branchwidth can be approximated within a factor of 2.25 [7] (a graph H is a *minor* of a graph G if H can be obtained by a subgraph of G after applying edge contractions). Finally, it follows from [13] that for every minor closed graph class, branchwidth can be approximated by a constant factor.

Branchwidth cannot increase when applying edge contractions or removals. According to the Graph Minors theory, this implies that, for any fixed k, there is a finite number of minor minimal graphs of branchwidth more than k and we denote this set of graphs by $\mathcal{B}_k$. Checking whether a graph G contains a fixed graph as a minor can be done in polynomial time [27]. Therefore, the knowledge of $\mathcal{B}_k$ implies the construction of a polynomial time algorithm for deciding whether $\text{branchwidth}(G) \leq k$, for any fixed k. Unfortunately $\mathcal{B}_k$ is known only for small values of k. In particular, $\mathcal{B}_0 = \{P_2\}, B_1 = \{P_4, K_3\}, \mathcal{B}_2 = \{K_4\}$ and $\mathcal{B}_3 = \{K_5, V_8, M_6, Q_3\}$ (here K_r is a clique on r vertices, P_r is a path on r edges, V_8 is the graph obtained by a cycle on 8 vertices if we connect all pairs of vertices with cyclic distance 4, M_6 is the octahedron, and Q_3 is the 3-dimensional cube). However, for any fixed k, one can construct a linear, on $n = |V(G)|$, algorithm that decides whether an input graph G has branchwidth $\leq k$ and, if so, outputs the corresponding branch decomposition (see [3]). In technical terms, this implies that the problem of asking, for a given graph G, whether $\text{branchwidth}(G) \leq k$, parameterized by k is fixed parameter tractable (i. e. belongs in the parameterized complexity class FPT). (See [12] for further references on parameterized algorithms and complexity.) The algorithm in [3] is complicated and uses the technique of characteristic sequences, which was also used in order to prove the analogous result for treewidth. For the particular cases where $k \leq 3$, simpler algorithms exist that use the "reduction rule" technique (see [4]). We stress that $\mathcal{B}_4$ remains unknown while several elements of it have been detected so far (including the dodecahedron and the icosahedron graphs). There is a number of algorithms that for a given k in time $2^{O(k)} \cdot n^{O(1)}$ either decide that the branchwidth of a given graph is at least k, or construct a branch decomposition of width $O(k)$ (see [26]). These results can be generalized to compute the branchwidth of matroids and even more general parameters.

An exact algorithm for branchwidth appeared in [14]. Its complexity is $O((2 \cdot \sqrt{3})^n \cdot n^{O(1)})$. The algorithm exploits special properties of branchwidth (see also [24]).

In contrast to treewidth, edge maximal graphs of given branchwidth are not so easy to characterize (for treewidth there are just k-trees, i. e. chordal graphs with all maximal cliques of size $k+1$). An algorithm for generating such graphs has been given in [23] and reveals several structural issues on this parameter.

It is known that a large number of graph theoretical problems can be solved in linear time when their inputs are restricted to graphs of small (i. e. fixed) treewidth or branchwidth (see [2]).



Branchwidth of Graphs, Figure 1
Example of a graph and its branch decomposition of width 3

Branchwidth appeared to be a useful tool in the design of exact subexponential algorithms on planar graphs and their generalizations. The basic idea behind this approach is very simple: Let $\mathcal{P}$ be a problem on graphs and $\mathcal{G}$ be a class of graphs such that

- For every graph $G \in \mathcal{G}$ of branchwidth at most ℓ, the problem $\mathcal{P}$ can be solved in time $2^{c \cdot \ell} \cdot n^{\mathcal{O}(1)}$, where c is a constant, and;
- For every graph $G \in \mathcal{G}$ on n vertices a branch decomposition (not necessarily optimal) of G of width at most $h(n)$ can be constructed in polynomial time, where $h(n)$ is a function.

Then for every graph $G \in \mathcal{G}$, the problem $\mathcal{P}$ can be solved in time $2^{c \cdot h(n)} \cdot n^{\mathcal{O}(1)}$. Thus, everything boils down to computations of constants c and functions $h(n)$. These computations can be quite involved. For example, as was shown in [17], for every planar graph G on n vertices, the branchwidth of G is at most $\sqrt{4.5n} < 2.1214\sqrt{n}$. For extensions of this bound to graphs embeddable on a surface of genus g, see [15].

Dorn [9] used fast matrix multiplication in dynamic programming to estimate the constants c for a number of problems. For example, for the MAXIMUM INDEPENDENT SET problem, $c \leq \omega/2$, where $\omega < 2.376$ is the matrix product exponent over a ring, which implies that the INDEPENDENT SET problem on planar graphs is solvable in time $O(2^{2.52\sqrt{n}})$. For the MINIMUM DOMINATING SET problem, $c \leq 4$, thus implying that the branch decomposition method runs in time $O(2^{3.99\sqrt{n}})$. It appears that algorithms of running time $2^{O(\sqrt{n})}$ can be designed even for some of the "non-local" problems, such as the HAMILTONIAN CYCLE, CONNECTED DOMINATING SET, and STEINER TREE, for which no time $2^{O(\ell)} \cdot n^{\mathcal{O}(1)}$ algorithm on general graphs of branchwidth ℓ is known [11]. Here one needs special properties of some optimal planar branch decompositions, roughly speaking that every edge of T corresponds to a disk on a plane such that all edges of G corresponding to one component of $T - e$ are inside the disk and all other edges are outside. Some of the subexponential algorithms on planar graphs can be generalized for graphs embedded on surfaces [10] and, more generally, to graph classes that are closed under taking of minors [8].

A similar approach can be used for parameterized problems on planar graphs. For example, a parameterized algorithm that finds a dominating set of size $\leq k$ (or reports that no such set exists) in time $2^{\mathcal{O}(\sqrt{k})} n^{\mathcal{O}(1)}$ can be obtained based on the following observations: there is a constant c such that every planar graph of branchwidth at least $c\sqrt{k}$ does not contain a dominating set of size at most k. Then for a given k the algorithm computes an optimal branch decomposition of a palanar graph G and if its width is more than $c\sqrt{k}$ concludes that G has no dominating set of size k. Otherwise, find an optimal dominating set by performing dynamic programming in time $2^{\mathcal{O}(\sqrt{k})} n^{\mathcal{O}(1)}$. There are several ways of bounding a parameter of a planar graph in terms of its branchwidth or treewidth including techniques similar to Baker's approach from approximation algorithms [1], the use of separators, or by some combinatorial arguments, as shown in [16]. Another general approach of bounding the branchwidth of a planar graph by parameters, is based on the results of Robertson et al. [28] regarding quickly excluding a planar graph. This brings us to the notion of *bidimensionality* [6]. Parameterized algorithms based on branch decompositions can be generalized from planar graphs to graphs embedded on surfaces and to graphs excluding a fixed graph as a minor.

Applications

See [5] for using branchwidth for solving TSP.

Open Problems

1. It is known that any planar graph G has branchwidth at most $\sqrt{4.5} \cdot \sqrt{|V(G)|}$ (or at most $\frac{3}{2} \cdot \sqrt{|E(G)|+2}$) [17]. Is it possible to improver this upper bound? Any possible improvement would accelerate many of the known exact or parameterized algorithms on planar graphs that use dynamic programming on branch decompositions.
2. In contrast to treewidth, very few graph classes are known where branchwidth is computable in polynomial time. Find graphs classes where branchwidth can be computed or approximated in polynomial time.
3. Find $\mathcal{B}_k$ for values of k bigger than 3. The only structural result on $\mathcal{B}_k$ is that its planar elements will be either self-dual or pairwise-dual. This follows from the fact that dual planar graphs have the same branchwidth [29,16].
4. Find an exact algorithm for branchwidth of complexity $O^*(2^n)$ (the notation $O^*()$ assumes that we drop the non-exponential terms in the classic $O()$ notation).
5. The dependence on k of the linear time algorithm for branchwidth in [3] is huge. Find an $2^{O(k)} \cdot n^{O(1)}$ step algorithm, deciding whether the branchwidth of an n-vertex input graph is at most k.

Cross References

► Bidimensionality
► Treewidth of Graphs

Recommended Reading

1. Alber, J., Bodlaender, H.L., Fernau, H., Kloks, T., Niedermeier, R.: Fixed parameter algorithms for dominating set and related problems on planar graphs. Algorithmica **33**, 461–493 (2002)
2. Arnborg, S.: Efficient algorithms for combinatorial problems on graphs with bounded decomposability – A survey. BIT **25**, 2–23 (1985)
3. Bodlaender, H.L., Thilikos, D.M.: Constructive linear time algorithms for branchwidth. In: Automata, languages and programming (Bologna, 1997). Lecture Notes in Computer Science, vol. 1256, pp. 627–637. Springer, Berlin (1997)
4. Bodlaender, H.L., Thilikos, D.M.: Graphs with branchwidth at most three. J. Algorithms **32**, 167–194 (1999)
5. Cook, W., Seymour, P.D.: Tour merging via branch-decomposition. Inf. J. Comput. **15**, 233–248 (2003)
6. Demaine, E.D., Fomin, F.V., Hajiaghayi, M., Thilikos, D.M.: Bidimensional parameters and local treewidth. SIAM J. Discret. Math. **18**, 501–511 (2004)
7. Demaine, E.D., Hajiaghayi, M.T., Nishimura, N., Ragde, P., Thilikos, D. M.: Approximation algorithms for classes of graphs excluding single-crossing graphs as minors. J. Comput. Syst. Sci. **69**, 166–195 (2004)
8. Dorn, F., Fomin, F.V., Thilikos, D.M.: Subexponential algorithms for non-local problems on H-minor-free graphs. In: Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms (SODA 2008). pp. 631–640. Society for Industrial and Applied Mathematics, Philadelphia (2006)
9. Dorn, F.: Dynamic programming and fast matrix multiplication. In: Proceedings of the 14th Annual European Symposium on Algorithms (ESA 2006). Lecture Notes in Computer Science, vol. 4168 , pp. 280–291. Springer, Berlin (2006)
10. Dorn, F., Fomin, F.V., Thilikos, D.M.: Fast subexponential algorithm for non-local problems on graphs of bounded genus. In: Proceedings of the 10th Scandinavian Workshop on Algorithm Theory (SWAT 2006). Lecture Notes in Computer Science. Springer, Berlin (2005)
11. Dorn, F., Penninkx, E., Bodlaender, H., Fomin, F.V.: Efficient exact algorithms on planar graphs: Exploiting sphere cut branch decompositions. In: Proceedings of the 13th Annual European Symposium on Algorithms (ESA 2005). Lecture Notes in Computer Science, vol. 3669, pp. 95–106. Springer, Berlin (2005)
12. Downey, R.G., Fellows, M.R.: Parameterized complexity. In: Monographs in Computer Science. Springer, New York (1999)
13. Feige, U., Hajiaghayi, M., Lee, J.R.: Improved approximation algorithms for minimum-weight vertex separators. In: Proceedings of the 37th annual ACM Symposium on Theory of computing (STOC 2005), pp. 563–572. ACM Press, New York (2005)
14. Fomin, F.V., Mazoit, F., Todinca, I.: Computing branchwidth via efficient triangulations and blocks. In: Proceedings of the 31st Workshop on Graph Theoretic Concepts in Computer Science (WG 2005). Lecture Notes Computer Science, vol. 3787, pp. 374–384. Springer, Berlin (2005)
15. Fomin, F.V., Thilikos, D.M.: Fast parameterized algorithms for graphs on surfaces: Linear kernel and exponential speed-up. In: Proceedings of the 31st International Colloquium on Automata, Languages and Programming (ICALP 2004). Lecture Notes Computer Science, vol. 3142, pp. 581–592. Springer, Berlin (2004)
16. Fomin, F.V., Thilikos, D.M.: Dominating sets in planar graphs: Branch-width and exponential speed-up. SIAM J. Comput. **36**, 281–309 (2006)
17. Fomin, F.V., Thilikos, D.M.: New upper bounds on the decomposability of planar graphs. J. Graph Theor. **51**, 53–81 (2006)
18. Gu, Q.P., Tamaki, H.: Optimal branch-decomposition of planar graphs in O(n^3) time. In: Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP 2005). Lecture Notes Computer Science, vol. 3580, pp. 373–384. Springer, Berlin (2005)
19. Gu, Q.P., Tamaki, H.: Branch-width, parse trees, and monadic second-order logic for matroids. J. Combin. Theor. Ser. B **96**, 325–351 (2006)
20. Kloks, T., Kratochvíl, J., Müller, H.: Computing the branchwidth of interval graphs. Discret. Appl. Math. **145**, 266–275 (2005)
21. Mazoit, F.: The branch-width of circular-arc graphs. In: 7th Latin American Symposium on Theoretical Informatics (LATIN 2006), 2006, pp. 727–736

22. Oum, S.I., Seymour, P.: Approximating clique-width and branch-width. J. Combin. Theor. Ser. B **96**, 514–528 (2006)
23. Paul, C., Proskurowski, A., Telle, J.A.: Generating graphs of bounded branchwidth. In: Proceedings of the 32nd Workshop on Graph Theoretic Concepts in Computer Science (WG 2006). Lecture Notes Computer Science, vol. 4271, pp. 205–216. Springer, Berlin (2006)
24. Paul, C., Telle, J.A.: New tools and simpler algorithms for branchwidth. In: Proceedings of the 13th Annual European Symposium on Algorithms (ESA 2005), 2005 pp. 379–390
25. Robertson, N. Seymour, P.D.: Graph minors. X. Obstructions to tree-decomposition J. Combin. Theor. Ser. B **52**, 153–190 (1991)
26. Robertson, N. Seymour, P.D.: Graph minors. XII. Distance on a surface. J. Combin. Theor. Ser. B **64**, 240–272 (1995)
27. Robertson, N. Seymour, P.D.: Graph minors. XIII. The disjoint paths problem. J. Combin. Theor. Ser. B **63**, 65–110 (1995)
28. Robertson, N., Seymour, P.D., Thomas, R.: Quickly excluding a planar graph. J. Combin. Theor. Ser. B **62**, 323–348 (1994)
29. Seymour, P.D., Thomas, R.: Call routing and the ratcatcher. Combinatorica **14**, 217–241 (1994)

Broadcasting in Geometric Radio Networks
2001; Dessmark, Pelc

ANDRZEJ PELC
Department of Computer Science,
University of Québec-Ottawa, Gatineau, QC, Canada

Keywords and Synonyms

Wireless information dissemination in geometric networks

Problem Definition

The Model Overview

Consider a set of stations (nodes) modeled as points in the plane, labeled by natural numbers, and equipped with transmitting and receiving capabilities. Every node u has a *range* r_u depending on the power of its transmitter, and it can reach all nodes at distance at most r_u from it. The collection of nodes equipped with ranges determines a directed graph on the set of nodes, called a *geometric radio network* (GRN), in which a directed edge (uv) exists if node v can be reached from u. In this case u is called a *neighbor* of v. If the power of all transmitters is the same then all ranges are equal and the corresponding GRN is symmetric.

Nodes send messages in synchronous *rounds*. In every round every node acts either as a *transmitter* or as a *receiver*. A node gets a message in a given round, if and only if, it acts as a receiver and exactly one of its neighbors transmits in this round. The message received in this case is the one that was transmitted. If at least two neighbors of a receiving node u transmit simultaneously in a given round, none of the messages is received by u in this round. In this case it is said that a *collision* occurred at u.

The Problem

Broadcasting is one of the fundamental network communication primitives. One node of the network, called the *source*, has to transmit a message to all other nodes. Remote nodes are informed via intermediate nodes, along directed paths in the network. One of the basic performance measures of a broadcasting scheme is the total time, i. e., the number of rounds it uses to inform all the nodes of the network.

For a fixed real $s \geq 0$, called the *knowledge radius*, it is assumed that each node knows the part of the network within the circle of radius s centered at it, i. e., it knows the positions, labels and ranges of all nodes at distance at most s. The following problem is considered:

How the size of the knowledge radius influences deterministic broadcasting time in GRN?

Terminology and Notation

Fix a finite set $R = \{r_1, \dots, r_\rho\}$ of positive reals such that $r_1 < \dots < r_\rho$. Reals r_i are called *ranges*. A *node* v is a triple $[l, (x, y), r_i]$, where l is a binary sequence called the *label* of v, (x, y) are coordinates of a point in the plane, called the *position* of v, and $r_i \in R$ is called the *range* of v. It is assumed that labels are consecutive integers 1 to n, where n is the number of nodes, but all the results hold if labels are integers in the set $\{1, \dots, M\}$, where $M \in O(n)$. Moreover, it is assumed that all nodes know an upper bound Γ on n, where Γ is polynomial in n. One of the nodes is distinguished and called the *source*. Any set of nodes C with a distinguished source, such that positions and labels of distinct nodes are different is called a *configuration*.

With any configuration C the following directed graph $\mathcal{G}(C)$ is associated. Nodes of the graph are nodes of the configuration and a directed edge (uv) exists in the graph, if and only if the distance between u and v does not exceed the range of u. (The word "distance" always means the geometric distance in the plane and not the distance in a graph.) In this case u is called a neighbor of v. Graphs of the form $\mathcal{G}(C)$ for some configuration C are called *geometric radio networks* (GRN). In what follows, only configurations C such that in $\mathcal{G}(C)$ there exists a directed path from the source to any other node, are considered. If the size of the set R of ranges is ρ, a resulting configuration and the corresponding GRN are called a ρ-configuration and

ρ-GRN, respectively. Clearly, all 1-GRN are symmetric graphs. D denotes the *eccentricity* of the source in a GRN, i. e., the maximum length of all shortest paths in this graph from the source to all other nodes. D is of order of the diameter if the graph is symmetric but may be much smaller in general. $\Omega(D)$ is an obvious lower bound on broadcasting time.

Given any configuration, fix a non-negative real s, called the *knowledge radius*, and assume that every node of C has initial input consisting of all nodes whose positions are at distance at most s from its own. Thus it is assumed that every node knows *a priori* labels, positions and ranges of all nodes within a circle of radius s centered at it. All nodes also know the set R of available ranges.

It is not assumed that nodes know any global parameters of the network, such as its size or diameter. The only global information that nodes have about the network is a polynomial upper bound on its size. Consequently, the broadcast process may be finished but no node needs to be aware of this fact. Hence the adopted definition of broadcasting time is the same as in [3]. An algorithm accomplishes broadcasting in t rounds, if all nodes know the source message after round t, and no messages are sent after round t.

Only deterministic algorithms are considered. Nodes can transmit messages even before getting the source message, which enables preprocessing in some cases. The algorithms are *adaptive*, i. e., nodes can schedule their actions based on their local history. A node can obviously gain knowledge from previously obtained messages. There is, however, another potential way of acquiring information during the communication process. The availability of this method depends on what happens during a collision, i. e., when u acts as a receiver and two or more neighbors of u transmit simultaneously. As mentioned above, u does not get any of the messages in this case. However, two scenarios are possible. Node u may either hear nothing (except for the background noise), or it may receive *interference noise* different from any message received properly but also different from background noise. In the first case it is said that there is no *collision detection*, and in the second case – that collision detection is available (cf., e. g., [1]). A discussion justifying both scenarios can be found in [1,7].

Related Work

Broadcasting in geometric radio networks and some of their variations was considered, e. g., in [6,8,10,11]. In [11] the authors proved that scheduling optimal broadcasting is NP-hard even when restricted to such graphs, and gave an $O(n \log n)$ algorithm to schedule an optimal broadcast when nodes are situated on a line. In [10] broadcasting was considered in networks with nodes randomly placed on a line. In [8] the authors discussed fault-tolerant broadcasting in radio networks arising from regular locations of nodes on the line and in the plane, with reachability regions being squares and hexagons, rather than circles. Finally, in [6] broadcasting with restricted knowledge was considered but the authors studied only the special case of nodes situated on the line.

Key Results

The results summarized below are based on the paper [5] of which [4] is a preliminary version.

Arbitrary GRN in the Model Without Collision Detection

Clearly all upper bounds and algorithms are valid in the model with collision detection as well.

Large Knowledge Radius

Theorem 1 *The minimum time to perform broadcasting in an arbitrary GRN with source eccentricity D and knowledge radius $s > r_\rho$ (or with global knowledge of the network) is $\Theta(D)$.*

This result yields a centralized $O(D)$ broadcasting algorithm when global knowledge of the GRN is available. This is in sharp contrast with broadcasting in arbitrary graphs, as witnessed by the graph from [9] which has bounded diameter but requires time $\Omega(\log n)$ for broadcasting.

Knowledge Radius Zero Next consider the case when knowledge radius $s = 0$, i. e., when every node knows only its own label, position and range. In this case it is possible to broadcast in time $O(n)$ for arbitrary GRN. It should be stressed that this upper bound is valid for arbitrary GRN, not only symmetric, unlike the algorithm from [3] designed for arbitrary symmetric graphs.

Theorem 2 *It is possible to broadcast in arbitrary n-node GRN with knowledge radius zero in time $O(n)$.*

The above upper bound for GRN should be contrasted with the lower bound from [2,3] showing that some graphs require broadcasting time $\Omega(n \log n)$. Indeed, the graphs constructed in [2,3] and witnessing to this lower bound are not GRN. Surprisingly, this sharper lower bound does not require very unusual graphs. While counterexamples from [2,3] are not GRN, it turns out that the reason for a longer broadcasting time is really not the topology of the graph but the difference in knowledge available to nodes.

Recall that in GRN with knowledge radius 0 it is assumed that each node knows its own position (apart from its label and range): the upper bound $O(n)$ uses this geometric information extensively.

If this knowledge is not available to nodes (i. e., each node knows only its label and range) then there exists a family of GRN requiring broadcasting time $\Omega(n \log n)$. Moreover it is possible to show such GRN resulting from configurations with only 2 distinct ranges. (Obviously for 1-configurations this lower bound does not hold, as these configurations yield symmetric GRN and in [3] the authors showed an $O(n)$ algorithm working for arbitrary symmetric graphs).

Theorem 3 *If every node knows only its own label and range (and does not know its position) then there exist n-node GRN requiring broadcasting time $\Omega(n \log n)$.*

Symmetric GRN

The Model with Collision Detection In the model with collision detection and knowledge radius zero optimal broadcast time is established by the following pair of results.

Theorem 4 *In the model with collision detection and knowledge radius zero it is possible to broadcast in any n-node symmetric GRN of diameter D in time $O(D + \log n)$.*

The next result is the lower bound $\Omega(\log n)$ for broadcasting time, holding for some GRN of diameter 2. Together with the obvious bound $\Omega(D)$ this matches the upper bound from Theorem 4.

Theorem 5 *For any broadcasting algorithm with collision detection and knowledge radius zero, there exist n-node symmetric GRN of diameter 2 for which this algorithm requires time $\Omega(\log n)$.*

The Model Without Collision Detection For the model without collision detection. it is possible to maintain complexity $O(D + \log n)$ of broadcasting. However, a stronger assumption concerning knowledge radius is needed: it is no longer 0, but positive, although arbitrarily small.

Theorem 6 *In the model without collision detection, it is possible to broadcast in any n-node symmetric GRN of diameter D in time $O(D + \log n)$, for any positive knowledge radius.*

Applications

The radio network model is applicable to wireless networks using a single frequency. The specific model of geometric radio networks described in Sect. "Problem Definition" is applicable to wireless networks where stations are located in a relatively flat region without large obstacles (natural or human made), e. g., in the sea or a desert, as opposed to a large city or a mountain region. In such a terrain, the signal of a transmitter reaches receivers at the same distance in all directions, i. e., the set of potential receivers of a transmitter is a disc.

Open Problems

1. Is it possible to broadcast in time $o(n)$ in arbitrary n-node GRN with eccentricity D sublinear in n, for knowledge radius zero?
   Note: in view of Theorem 2 it is possible to broadcast in time $O(n)$.
2. Is it possible to broadcast in time $O(D + \log n)$ in all symmetric n-node GRN with eccentricity D, without collision detection, when knowledge radius is zero?
   Note: in view of Theorems 4 and 6, the answer is positive if either collision detection or a positive (even arbitrarily small) knowledge radius is assumed.

Cross References

► Deterministic Broadcasting in Radio Networks
► Randomized Broadcasting in Radio Networks
► Randomized Gossiping in Radio Networks
► Routing in Geometric Networks

Acknowledgments

Research partially supported by NSERC discovery grant and by the Research Chair in Distributed Computing at the Université du Québec en Outaouais.

Recommended Reading

1. Bar-Yehuda, R., Goldreich, O., Itai, A.: On the time complexity of broadcast in radio networks: an exponential gap between determinism and randomization. J. Comput. Syst. Sci. **45**, 104–126 (1992)
2. Bruschi, D., Del Pinto, M.: Lower bounds for the broadcast problem in mobile radio networks. Distrib. Comput. **10**, 129–135 (1997)
3. Chlebus, B.S., Gasieniec, L., Gibbons, A., Pelc, A., Rytter, W.: Deterministic broadcasting in ad hoc radio networks. Distrib. Comput. **15**, 27–38 (2002)
4. Dessmark, A., Pelc, A.: Tradeoffs between knowledge and time of communication in geometric radio networks. Proc. 13th Ann. ACM Symposium on Parallel Algorithms and Architectures (SPAA), pp. 59–66, Crete Greece, July 3–6, 2001
5. Dessmark, A., Pelc, A.: Broadcasting in geometric radio networks. J. Discret. Algorithms **5**, 187–201 (2007)

6. Diks, K., Kranakis, E., Krizanc, D., Pelc, A.: The impact of knowledge on broadcasting time in linear radio networks. Theor. Comput. Sci. **287**, 449–471 (2002)
7. Gallager, R.: A Perspective on Multiaccess Channels. IEEE Trans. Inf. Theory **31**, 124–142 (1985)
8. Kranakis, E., Krizanc, D., Pelc, A.: Fault-tolerant broadcasting in radio networks. J. Algorithms **39**, 47–67 (2001)
9. Pelc, A., Peleg, D.: Feasibility and complexity of broadcasting with random transmission failures. Proc. 24th Ann. ACM Symposium on Principles of Distributed Computing (PODC), pp. 334–341, Las Vegas, July 17–20 2005
10. Ravishankar, K., Singh, S.: Broadcasting on [0, *L*]. Discret. Appl. Math. **53**, 299–319 (1994)
11. Sen, A., Huson, M. L.: A New Model for Scheduling Packet Radio Networks. Proc. 15th Annual Joint Conference of the IEEE Computer and Communication Societies (IEEE INFOCOM'96), pp. 1116–1124, San Francisco, 24–28 March, 1996

B-trees

1972; Bayer, McCreight

Jan Vahrenhold
Faculty of Computer Science,
Dortmund University of Technology,
Dortmund, Germany

Keywords and Synonyms

Multiway Search Trees

Problem Definition

This problem is concerned with storing a linearly ordered set of elements such that the Dictionary operations Find, Insert, and Delete can be performed efficiently.

In 1972, Bayer and McCreight introduced the class of B-trees as a possible way of implementing an "index for a dynamically changing random access file" [6, p. 173]. B-trees have received considerable attention both in the database and in the algorithms community ever since; a prominent witness to their immediate and widespread acceptance is the fact that the authoritative survey on B-trees authored by Comer [9] appeared as soon as 1979 and, already at that time, referred to the B-tree data structure as the "ubiquitous B-tree".

Notations

A *B-tree* is a multiway search tree defined as follows (the definition of Bayer and McCreight [6] is restated according to Knuth [16, Sect. 6.2.4] and Cormen et al. [10, Chap. 18.1]):

Definition 1 Let $m \geq 3$ be a positive integer. A tree T is a *B-tree* of degree m if it is either empty or fulfills the following properties:

1. All leaves of T appear on the same level of T.
2. Every node of T has at most m children.
3. Every node of T, except for the root and the leaves, has at least $m/2$ children.
4. The root of T is either a leaf or has at least two children.
5. An internal node with k children $c_1[\nu], \dots, c_k[\nu]$ stores $k-1$ keys, and a leaf stores between $m/2-1$ and $m-1$ keys. The keys $\text{key}_i[\nu]$, $1 \leq i \leq k-1$, of a node $\nu \in T$ are maintained in sorted order, i. e. $\text{key}_1[\nu] \leq \dots \leq \text{key}_{k-1}[\nu]$.
6. If ν is an internal node of T with k children $c_1[\nu], \dots, c_k[\nu]$, the $k-1$ keys $\text{key}_1[\nu], \dots, \text{key}_{k-1}[\nu]$ of ν separate the range of keys stored in the subtrees rooted at the children of ν. If x_i is any key stored in the subtree rooted at $c_i[\nu]$, the following holds:

$$x_1 \leq \text{key}_1[\nu] \leq x_2 \leq \text{key}_2[\nu] \leq \dots \\ \leq x_{k-1} \leq \text{key}_{k-1}[\nu] \leq x_k \, .$$

To search a B-tree for a given key x, the algorithm starts with the root of the tree being the current node. If x matches one of the current node's keys, the search terminates successfully. Otherwise, if the current node is a leaf, the search terminates unsuccessfully. If the current node's keys do not contain x and if the current node is not a leaf, the algorithm identifies the unique subtree rooted at child of the current node that may contain x and recurses on this subtree. Since the keys of a node guide the search process, they are also referred to as *routing elements*.

Variants and Extensions

Knuth [16] defines a B^**-tree* to be a B-tree where Property 3 in Definition 1 is modified such that every node (except for the root) contains at least $2m/3$ keys.

A B^+*-tree* is a leaf-oriented B-tree, i. e. a B-tree that stores the keys in the leaves only. Additionally, the leaves are linked in left-to-right order to allow for fast sequential traversal of the keys stored in the tree. In a leaf-oriented tree, the routing elements usually are copies of certain keys stored in the leaves ($\text{key}_i[\nu]$ can be set to be the largest key stored in the subtree rooted at $c_i[\nu]$), but any set of routing elements that fulfills Properties 5 and 6 of Definition 1 can do as well.

Huddleston and Mehlhorn [13] extended Definition 1 to describe a more general class of multiway search trees that includes the class of B-trees as a special case. Their class of so-called (a, b)*-trees* is parametrized by two integers a and b with $a \geq 2$ and $2a-1 \leq b$. Property 2 of

Definition 1 is modified to allow each node to have up to b children and Property 3 is modified to require that, except for the root and the leaves, every node of an (a, b)-tree has at least a children. All other properties of Definition 1 remain unchanged for (a, b)-trees. Usually, (a, b)-trees are implemented as leaf-oriented trees.

By the above definitions, a B-tree is a $(b/2, b)$-tree (if b is even) or an $(a, 2a-1)$-tree (if b is odd). The subtle difference between even and odd maximum degree becomes relevant in an important amortization argument of Huddleston and Mehlhorn (see below) where the inequality $b \geq 2a$ is required to hold. This amortization argument actually caused (a, b)-trees with $b \geq 2a$ to be given a special name: *weak* B-trees [13].

Update Operations

An INSERT operation on an (a, b)-tree first tries to locate the key x to be inserted. After an unsuccessful search that stops at some leaf ℓ, x is inserted into ℓ's set of keys. If ℓ becomes too full, i. e. contains more than b elements, two approaches are possible to resolve this *overflow* situation: (1) the node ℓ can be split around its median key into two nodes with at least a keys each or (2) the node ℓ can have some of its keys be distributed to its left or right siblings (if this sibling has enough space to accommodate the new keys). In the first case, a new routing element separating the keys in the two new subtrees of ℓ's parent μ has to be inserted into the key set of μ, and in the second case, the routing element in μ separating the keys in the subtree rooted at ℓ from the keys rooted at ℓ's relevant sibling needs to be updated. If ℓ was split, the node μ needs to be checked for a potential overflow due to the insertion of a new routing element, and the split may propagate all the way up to the root.

A DELETE operation also first locates the key x to be deleted. If (in a non-leaf-oriented tree) x resides in an internal node, x is replaced by the largest key in the left subtree of x (or the smallest key in the right subtree of x) which resides in a leaf and is deleted from there. In a leaf-oriented tree, keys are deleted from leaves only (the correctness of a routing element on a higher levels is not affected by this deletion). In any case, a DELETE operation may result in a leaf node ℓ containing less than a elements. Again, there are two approaches to resolve this *underflow* situation: (1) the node ℓ is merged with its left or right sibling node or (2) keys from ℓ's left or right sibling node are moved to ℓ (unless the sibling node would underflow as a result of this). Both underflow handling strategies require updating the routing information stored in the parent of ℓ which (in the case of merging) may underflow itself. As with overflow handling, this process may propagate up to the root of the tree.

Note that the root of the tree can be split as a result of an INSERT operation and that it may disappear if the only two children of the root are merged to form the new root. This implies that B-trees grow and shrink at the top, and thus all leaves a guaranteed to appear on the same level of the tree (Property 1 of Definition 1).

Key Results

Since B-trees are the premier index structure for external storage, the results given in this section are stated not only in the RAM-model of computation but also in the I/O-model of computation introduced by Aggarwal and Vitter [1]. In the I/O-model, not only the number N of elements in the problem instance, but also the number M of elements that simultaneously can be kept in main memory and the number B of elements that fit into one disk block are (non-constant) parameters, and the complexity measure is the number of I/O-operations needed to solve a given problem instance. If B-trees are used in an external-memory setting, the degree m of the B-tree is usually chosen such that one node fits into one disk block, i. e., $m \in \Theta(B)$, and this is assumed implicitly whenever the I/O-complexity of B-trees is discussed.

Theorem 1 *The height of an N-key B-tree of degree* $m \geq 3$ *is bounded by* $\log_{\lceil m/2 \rceil}((N+1)/2)$.

Theorem 2 ([18]) *The storage utilization for large B-trees of high order under random insertions and deletions is approximately* $\ln 2 \approx 69\%$.

Theorem 3 *A B-tree may be used to implement the abstract data type* Dictionary *such that the operations* Find, Insert, *and* Delete *on a set of N elements from a linearly ordered domain can be performed in* $\mathcal{O}(\log N)$ *time (with* $\mathcal{O}(\log_B N)$ *I/O-operations) in the worst case.*

Remark 1 By threading the nodes of a B-tree, i. e. by linking the nodes according to their in-order traversal number, the operations PREV and NEXT can be performed in constant time (with a constant number of I/O-operations).

A (one-dimensional) *range query* asks for all keys that fall within a given query range (interval).

Lemma 1 *A B-tree supports (one-dimensional) range queries with* $\mathcal{O}(\log N + K)$ *time complexity* ($\mathcal{O}(\log_B N + K/B)$ *I/O-complexity) in the worst case where K is the number of keys reported.*

Under the convention that each update to a B-tree results in a new "version" of the B-tree, a *multiversion* B-tree is a B-tree that allows for updates of the current version but also supports queries in earlier versions.

Theorem 4 ([8]) *A multiversion B-tree can be constructed from a B-tree such that it is optimal with respect to the worst-case complexity of the* Find, Insert, *and* Delete *operations as well as to the worst-case complexity of answering range queries.*

Applications

Databases

One of the main reasons for the success of the B-tree lies in its close connection to databases: any implementation of Codd's relational data model (introduced incidentally in the same year as B-trees were invented) requires an efficient indexing mechanism to search and traverse relations that are kept on secondary storage. If this index is realized as a B^+-tree, all keys are stored in a linked list of leaves which is indexed by the top levels of the B^+-tree, and thus both efficient logarithmic searching and sequential scanning of the set of keys is possible.

Due to the importance of this indexing mechanism, a wide number of results on how to incorporate B-trees and their variants into database systems and how to formulate algorithms using these structures have be published in the database community. Comer [9] and Graefe [12] summarize early and recent results but due to the bulk of results even these summaries cannot be fully comprehensive. Also, B-trees have been shown to work well in the presence of concurrent operations [7], and Mehlhorn [17, p. 212] notes that they perform especially well if a top-down splitting approach is used. The details of this splitting approach may be found, e. g., in the textbook of Cormen et al. [10, Chap. 18.2].

Priority Queues

A B-tree may be used to serve as an implementation of the abstract data type PRIORITYQUEUE since the smallest key always resides in the first slot of the leftmost leaf.

Lemma 2 *An implementation of a priority queue that uses a B-tree supports the* Min *operation in $\mathcal{O}(1)$ time (with $\mathcal{O}(1)$ I/O-operations). All other operations (including* DecreaseKey*) have a time complexity of $\mathcal{O}(\log N)$ (an I/O-complexity of $\mathcal{O}(\log_B N)$) in the worst case.*

Mehlhorn [17, Sect. III, 5.3.1] examined B-trees (and, more general, (a,b)-trees with $a \geq 2$ and $b \geq 2a-1$) in the context of *mergeable* priority queues. *Mergeable priority queues* are priority queues that additionally allow for concatenating and splitting priority queues. Concatenating priority queues for a set $S_1 \neq \emptyset$ and a set $S_2 \neq \emptyset$ is only defined if $\max\{x \mid x \in S_1\} < \min\{x \mid x \in S_2\}$ and results in a single priority queue for $S_1 \cup S_2$. Splitting a priority queue for a set $S_3 \neq \emptyset$ according to some $y \in \operatorname{dom}(S_3)$ results in a priority queue for the set $S_4 := \{x \in S_3 \mid x \leq y\}$ and a priority queue for the set $S_5 := \{x \in S_3 \mid x > y\}$ (one of these sets may be empty). Mehlhorn's result restated in the context of B-trees is as follows:

Theorem 5 (Theorem 6, Sect. III, 5.3.1 in [7]) *If sets $S_1 \neq \emptyset$ and $S_2 \neq \emptyset$ are represented by a B-tree each then operation* Concatenate(S_1, S_2) *takes time $\mathcal{O}(\log \max\{|S_1|, |S_2|\})$ (has an I/O-complexity of $\mathcal{O}(\log_B \max\{|S_1|, |S_2|\})$) and operation* Split$(S_1, y)$ *takes time $\mathcal{O}(\log |S_1|)$ (has an I/O-complexity of $\mathcal{O}(\log_B |S_1|)$). All bounds hold in the worst case.*

Buffered Data Structures

Many applications (including sorting) that involve massive data sets allow for batched data processing. A variant of B-trees that exploits this relaxed problem setting is the so-called *buffer tree* proposed by Arge [3]. A *buffer tree* is a B-trees of degree $m \in \Theta(M/B)$ (instead of $m \in \Theta(B)$) where each node is assigned a buffer of size $\Theta(M)$. These buffers are used to collect updates and query requests that are passed further down the tree only if the buffer gets full enough to allow for cost amortization.

Theorem 6 (Theorem 1 in [3]) *The total cost of an arbitrary sequence of N intermixed* Insert *and* Delete *operations on an initially empty buffer tree is $\mathcal{O}(N/B \log_{M/B} N/B)$ I/O operations, that is the amortized I/O-cost of an operation is $\mathcal{O}(1/B \log_{M/B} N/B)$.*

As a consequence, N elements can be sorted spending an optimal number of $\mathcal{O}(N/B \log_{M/B} N/B)$ I/O-operations by inserting them into a (leaf-oriented) buffer tree in a batched manner and then traversing the leaves. By the preceding discussion, buffer trees can also be used to implement (batched) priority queues in the external memory setting. Arge [3] extended his analysis of buffer trees to show that they also support DELETEMIN operations with an amortized I/O-cost of $\mathcal{O}(1/B \log_{M/B} N/B)$.

Since the degree of a buffer tree is too large to allow for efficient single-shot, i. e. non-batched operations, Arge et al. [4] discussed how buffers can be attached to (and later detached from) a multiway tree while at the same time keeping the degree of the base structure in $\Theta(B)$. Their discussion uses the R-tree index structure as a running example, the techniques presented, however, carry over to the B-tree. The resulting data structure is accessed through standard methods and additionally allows for batched update operations, e. g. *bulk loading*, and queries. The amortized I/O-complexity of all operations is analogous to the complexity of the buffer tree operations.

B-trees as Base Structures

Several external memory data structures are derived from B-trees or use a B-tree as their base structure—see the survey by Arge [2] for a detailed discussion. One of these structures, the so-called *weight-balanced* B-tree is particularly useful as a base tree for building dynamic external data structures that have secondary structures attached to all (or some) of their nodes. The weight-balanced B-tree, developed by Arge and Vitter [5], is a variant of the B-tree that requires all subtrees of a node to have approximately, i. e., up to a small constant factor, the same number of leaves. Weight-balanced B-trees can be shown to have the following property:

Theorem 7 ([5]) *In a weight-balanced B-tree, rebalancing after an update operation is performed by splitting or merging nodes. When a rebalancing operation involves a node v that is the root of a subtree with $w(v)$ leaves, at least $\Theta(w(v))$ update operations involving leaves below v have to be performed before v itself has to be rebalanced again.*

Using the above theorem, amortized bounds for maintaining secondary data structures attached to nodes of the base tree can be obtained—as long as each such structure can be updated with an I/O-complexity linear in the number of elements stored below the node it is attached to [2,5].

Amortized Analysis

Most of the amortization arguments used for (a, b)-trees, buffer trees, and their relatives are based upon a theorem due to Huddleston and Mehlhorn [13, Theorem 3]. This theorem states that the total number of rebalancing operations in any sequence of N intermixed insert and delete operations performed on an initially empty *weak* B-tree, i. e. an (a, b)-tree with $b \geq 2a$, is at most linear in N. This result carries over to buffer trees since they are $(M/4B, M/B)$-trees. Since B-trees are (a, b)-trees with $b = 2a - 1$ (if b is odd), the result in its full generality is not valid for B-trees, and Huddleston and Mehlhorn present a simple counterexample for $(2, 3)$-trees.

A crucial fact used in the proof of the above amortization argument is that the sequence of operations to be analyzed is performed on an initially *empty* data structure. Jacobsen et al. [14] proved the existence of *non-extreme* (a, b)-trees, i. e. (a, b)-trees where only few nodes have a degree of a or b. Based upon this, they re-established the above result that the rebalancing cost in a sequence of operations is amortized constant (and thus the related result for buffer trees) also for operations on initially non-empty data structures.

In connection with concurrent operations in database systems, it should be noted that the analysis of Huddleston and Mehlhorn actually requires $b \geq 2a + 2$ if a top-down splitting approach is used. In can be shown, though, that even in the general case, few node splits (in an amortized sense) happen close to the root.

URL to Code

There is a variety of (commercial and free) implementations of B-trees and (a, b)-trees available for download. Representatives are the C++-based implementations that are part of the LEDA-library (http://www.algorithmic-solutions.com), the STXXL-library (http://stxxl.sourceforge.net), and the TPIE-library (http://www.cs.duke.edu/TPIE) as well as the `Java`-based implementation that is part of the `javaxxl`-library (http://www.xxl-library.de). Furthermore, (pseudo-code) implementations can be found in almost every textbook on database systems or on algorithms and data structures—see, e. g., [10,11]. Since textbooks almost always leave developing the implementation details of the DELETE operation as an exercise to the reader, the discussion by Jannink [15] is especially helpful.

Cross References

- ► Cache-Oblivious B-Tree
- ► I/O-model
- ► R-Trees

Recommended Reading

1. Aggarwal, A., Vitter, J.S.: The input/output complexity of sorting and related problems. Commun. ACM **31**, 1116–1127 (1988)
2. Arge, L.A.: External memory data structures. In: Abello, J., Pardalos, P.M., Resende, M.G.C. (eds.) Handbook of Massive Data Sets, pp. 313–357. Kluwer, Dordrecht (2002)
3. Arge, L.A.: The Buffer Tree: A technique for designing batched external data structures. Algorithmica **37**, 1–24 (2003)
4. Arge, L.A., Hinrichs, K.H., Vahrenhold, J., Vitter, J.S.: Efficient bulk operations on dynamic R-trees. Algorithmica **33**, 104–128 (2002)
5. Arge, L.A., Vitter, J.S.: Optimal external interval management. SIAM J. Comput. **32**, 1488–1508 (2003)
6. Bayer, R., McCreight, E.M.: Organization and maintenance of large ordered indexes. Acta Inform. **1**, 173–189 (1972)
7. Bayer, R., Schkolnick, M.: Concurrency of operations on B-trees. Acta Inform. **9**, 1–21 (1977)

8. Becker, B., Gschwind, S., Ohler, T., Seeger, B., Widmayer, P.: An asymptotically optimal multiversion B-tree. VLDB J. **5**, 264–275 (1996)
9. Comer, D.E.: The ubiquitous B-tree. ACM Comput. Surv. **11**, 121–137 (1979)
10. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms. The MIT Electrical Engineering and Computer Science Series, 2nd edn. MIT Press, Cambridge (2001)
11. Elmasri, R., Navanthe, S.B.: Fundamentals of Database Systems, 5th edn. Addison-Wesley, Boston (2007)
12. Graefe, G.: B-tree indexes for high update rates. SIGMOD RECORD **35**, 39–44 (2006)
13. Huddleston, S., Mehlhorn, K.: A new data structure for representing sorted lists. Acta Inform. **17**, 157–184 (1982)
14. Jacobsen, L., Larsen, K.S., Nielsen, M.N.: On the existence of non-extreme (a, b)-trees. Inform. Process. Lett. **84**, 69–73 (2002)
15. Jannink, J.: Implementing deletions in B^+-trees. SIGMOD RECORD **24**, 33–38 (1995)
16. Knuth, D.E.: Sorting and Searching. The Art of Computer Programming, vol. 3, 2nd edn. Addison-Wesley, Reading (1998)
17. Mehlhorn, K.: Data Structures and Algorithms 1: Sorting and Searching. EATCS Monographs on Theoretical Computer Science, vol. 1. Springer, Berlin (1984)
18. Yao, A.C.-C.: On random 2–3 trees. Acta Inform. **9**, 159–170 (1978)

Buffering

► Direct Routing Algorithms
► Packet Switching in Single Buffer

Burrows–Wheeler Transform

1994; Burrows, Wheeler

PAOLO FERRAGINA[1], GIOVANNI MANZINI[2]
[1] Department of Computer Science, University of Pisa, Pisa, Italy
[2] Department of Computer Science, University of Eastern Piedmont, Alessandria, Italy

Keywords and Synonyms

Block-sorting data compression

Problem Definition

The Burrows–Wheeler transform is a technique used for the lossless compression of data. It is the algorithmic core of the tool bzip2 which has become a standard for the creation and distribution of compressed archives.

Before the introduction of the Burrows–Wheeler transform, the field of lossless data compression was dominated by two approaches (see [1,15] for comprehensive surveys). The first approach dates back to the pioneering works of Shannon and Huffman, and it is based on the idea of using shorter codewords for the more frequent symbols. This idea has originated the techniques of Huffman and Arithmetic Coding, and, more recently, the PPM (Prediction by Partial Matching) family of compression algorithms. The second approach originated from the works of Lempel and Ziv and is based on the idea of adaptively building a dictionary and representing the input string as a concatenation of dictionary words. The best-known compressors based on this approach form the so-called ZIP-family; they have been the standard for several years and are available on essentially any computing platform (e. g. gzip, zip, winzip, just to cite a few).

The Burrows–Wheeler transform introduced a completely new approach to lossless data compression based on the idea of *transforming* the input to make it easier to compress. In the authors' words: "(this) technique [. . .] works by applying a reversible transformation to a block of text to make redundancy in the input more accessible to simple coding schemes" [3, Sect. 7]. Not only has this technique produced some state-of-the-art compressors, but it also originated the field of compressed indexes [14] and it has been successfully extended to compress (and index) structured data such as XML files [7] and tables [16].

Key Results

Notation

Let s be a string of length n drawn from an alphabet Σ. For $i = 0, \dots, n-1$, $s[i]$ denotes the i-th character of s, and $s[i, n-1]$ denotes the suffix of s starting at position i (that is, starting with the character $s[i]$). Given two strings s and t, the notation $s \prec t$ is used to denote that s lexicographically precedes t.

The Burrows–Wheeler Transform

In [3] Burrows and Wheeler introduced a new compression algorithm based on a reversible transformation, now called the *Burrows–Wheeler Transform* (bwt). Given a string s, the computation of $\mathsf{bwt}(s)$ consists of three basic steps (see Fig. 1):

1. Append to the end of s a special symbol \$ smaller than any other symbol in Σ;
2. Form a *conceptual* matrix $\mathcal{M}$ whose rows are the cyclic shifts of the string $s\$$ sorted in lexicographic order;
3. Construct the transformed text $\hat{s} = \mathsf{bwt}(s)$ by taking the last column of $\mathcal{M}$.

Notice that every column of $\mathcal{M}$, hence also the transformed text $\hat{s}$, is a permutation of $s\$$. As an example

```
mississippi$                 $ mississipp i
ississippi$m                 i $mississip p
ssissippi$mi                 i ppi$missis s
sissippi$mis                 i ssippi$mis s
issippi$miss                 i ssissippi$ m
ssippi$missi         ⟹       m ississippi $
sippi$missis                 p i$mississi p
ippi$mississ                 p pi$mississ i
ppi$mississi                 s ippi$missi s
pi$mississip                 s issippi$mi s
i$mississipp                 s sippi$miss i
$mississippi                 s sissippi$m i
```

Burrows–Wheeler Transform, Figure 1
Example of Burrows–Wheeler transform for the string s=`mississippi`. The matrix on the right has the rows sorted in lexicographic order. The output of the bwt is the last column of the sorted matrix; in this example the output is $\hat{s}$ = bwt(s) = `ipssm$pissii`

F, the first column of the bwt matrix $\mathcal{M}$, consists of all characters of s alphabetically sorted. In Fig. 1 it is F = `$iiiimppssss`.

Although it is not obvious from its definition, the bwt is an invertible transformation and both the bwt and its inverse can be computed in $O(n)$ optimal time. To be consistent with the more recent literature, the following notation and proof techniques will be slightly different from the ones in [3].

Definition 1 For $1 \le i \le n$, let $s[k_i, n-1]$ denote the suffix of s prefixing row i of $\mathcal{M}$, and define $\Psi(i)$ as the index of the row prefixed by $s[k_i+1, n-1]$.

For example, in Fig. 1 it is $\Psi(2) = 7$ since row 2 of $\mathcal{M}$ is prefixed by `ippi` and row 7 is prefixed by `ppi`. Note that $\Psi(i)$ is not defined for $i = 0$ since row 0 is not prefixed by a proper suffix of s.[1]

Lemma 1 *For $i = 1, \ldots, n$, it is* $\mathsf{F}[i] = \hat{s}[\Psi(i)]$.

Proof Since each row contains a cyclic shift of $s\$$, the last character of the row prefixed by $s[k_i+1, n-1]$ is $s[k_i]$. Definition 1 then implies $\hat{s}[\Psi(i)] = s[k_i] = \mathsf{F}[i]$ as claimed. □

Lemma 2 *If $1 \le i < j \le n$ and* $\mathsf{F}[i] = \mathsf{F}[j]$ *then $\Psi(i) < \Psi(j)$.*

Proof Let $s[k_i, n-1]$ (resp. $s[k_j, n-1]$) denote the suffix of s prefixing row i (resp. row j). The hypothesis $i < j$ implies that $s[k_i, n-1] \prec s[k_j, n-1]$. The hypothesis $\mathsf{F}[i] = \mathsf{F}[j]$ implies $s[k_i] = s[k_j]$ hence it must be $s[k_i+1, n-1] \prec s[k_j+1, n-1]$. The thesis follows since by construction $\Psi(i)$ (resp. $\Psi(j)$) is the lexicographic position of the row prefixed by $s[k_i+1, n-1]$ (resp. $s[k_j+1, n-1]$). □

Lemma 3 *For any character $c \in \Sigma$, if* $\mathsf{F}[j]$ *is the ℓ-th occurrence of c in* F*, then $\hat{s}[\Psi(j)]$ is the ℓ-th occurrence of c in $\hat{s}$.*

Proof Take an index h such that $h < j$ and $\mathsf{F}[h] = \mathsf{F}[j] = c$ (the case $h > j$ is symmetric). Lemma 2 implies $\Psi(h) < \Psi(j)$ and Lemma 1 implies $\hat{s}[\Psi(h)] = \hat{s}[\Psi(j)] = c$. Consequently, the number of c's preceding (resp. following) $\mathsf{F}[j]$ in F coincides with the number of c's preceding (resp. following) $\hat{s}[\Psi(j)]$ in $\hat{s}$ and the lemma follows. □

In Fig. 1 it is $\Psi(2) = 7$ and both $\mathsf{F}[2]$ and $\hat{s}[7]$ are the second `i` in their respective strings. This property is usually expressed by saying that corresponding characters maintain the *same relative order* in both strings F and $\hat{s}$.

Lemma 4 *For any i, $\Psi(i)$ can be computed from $\hat{s}$ =* bwt(s).

Proof Retrieve F simply by sorting alphabetically the symbols of $\hat{s}$. Then compute $\Psi(i)$ as follows: (1) set $c = \mathsf{F}[i]$, (2) compute ℓ such that $\mathsf{F}[i]$ is the ℓ-th occurrence of c in F, (3) return the index of the ℓ-th occurrence of c in $\hat{s}$. □

Referring again to Fig. 1, to compute $\Psi(10)$ it suffices to set $c = \mathsf{F}[10] =$ `s` and observe that $\mathsf{F}[10]$ is the second `s` in F. Then it suffices to locate the index j of the second `s` in $\hat{s}$, namely $j = 4$. Hence $\Psi(10) = 4$, and in fact row 10 is prefixed by `sissippi` and row 4 is prefixed by `issippi`.

Theorem 5 *The original string s can be recovered from* bwt(s).

Proof Lemma 4 implies that the column F and the map Ψ can be retrieved from bwt(s). Let j_0 denote the index of the special character \$ in $\hat{s}$. By construction, the row j_0 of the bwt matrix is prefixed by $s[0, n-1]$, hence $s[0] = \mathsf{F}[j_0]$. Let $j_1 = \Psi(j_0)$. By Definition 1 row j_1 is prefixed by $s[1, n-1]$ hence $s[1] = \mathsf{F}[j_1]$. Continuing in this way it is straightforward to prove by induction that $s[i] = \mathsf{F}[\Psi^i(j_0)]$, for $i = 1, \ldots, n-1$. □

Algorithmic Issues

A remarkable property of the bwt is that both the direct and the inverse transform admit efficient algorithms that are extremely simple and elegant.

[1] In [3] instead of Ψ the authors make use of a map which is essentially the inverse of Ψ. The use of Ψ has been introduced in the literature of compressed indexes where Ψ and its inverse play an important role (see [14]).

Procedure sa2bwt
```
1. bwt[0]=s[n-1];
2. for(i=1;i<=n;i++)
3. if(sa[i] == 1)
4. bwt[i]='$';
5. else
6. bwt[i]=s[sa[i]-1];
```

Procedure bwt2psi
```
71. for(i=0;i<=n;i++)
2. c = bwt[i];
3. if(c == '$')
4. j0 = i;
5. else
6. h = count[c]++;
7. psi[h]=i;
```

Procedure psi2text
```
891. k = j0; i=0;
2. do
3. k = psi[k];
4. s[i++] = bwt[k];
   while(i<n);
```

Burrows–Wheeler Transform, Figure 2
Algorithms for computing and inverting the Burrows–Wheeler Transform. Procedure sa2bwt computes bwt(s) given s and its suffix array sa. Procedure bwt2psi takes bwt(s) as input and computes the Ψ map storing it in the array `psi`. bwt2psi also stores in `j0` the index of the row prefixed by $s[0, n-1]$. bwt2psi uses the auxiliary array `count`$[1, |\Sigma|]$ which initially contains in `count[i]` the number of occurrences in bwt(s) of the symbols $1, \dots, i-1$. Finally, procedure psi2text recovers the string s given bwt(s), the array `psi`, and the value j_0

Theorem 6 *Let $s[1, n]$ be a string over a constant size alphabet Σ. String $\hat{s} = \mathsf{bwt}(s)$ can be computed in $O(n)$ time using $O(n \log n)$ bits of working space.*

Proof The Suffix Array of s can be computed in $O(n)$ time and $O(n \log n)$ bits of working space by using, for example, the algorithm in [11]. The Suffix Array is an array of integers $\mathsf{sa}[1, n]$ such that for $i = 1, \dots, n$, $s[\mathsf{sa}[i], n-1]$ is the i-th suffix of s in the lexicographic order. Since each row of $\mathcal{M}$ is prefixed by a unique suffix of s followed by the special symbol \$, the Suffix Array provides the ordering of the rows in $\mathcal{M}$. Consequently, bwt(s) can be computed from sa in linear time using the procedure sa2bwt of Fig. 2. □

Theorem 7 *Let $s[1, n]$ be a string over a constant size alphabet Σ. Given* bwt(s)*, the string s can be retrieved in $O(n)$ time using $O(n \log n)$ bits of working space.*

Proof The algorithm for retrieving s follows almost verbatim the procedure outlined in the proof of Theorem 5. The only difference is that, for efficiency reasons, all the values of the map Ψ are computed in one shot. This is done by the procedure bwt2psi in Fig. 2. In bwt2psi instead of working with the column F, it uses the array count which is a "compact" representation of F. At the beginning of the procedure, for any character $c \in \Sigma$, count$[c]$ provides the index of the first row of $\mathcal{M}$ prefixed by c. For example, in Fig. 1 count[`i`] = 1, count[`m`] = 5, and so on. In the main `for` loop of bwt2psi the array bwt is scanned and count$[c]$ is increased every time an occurrence of character c is encountered (line 6). Line 6 also assigns to `h` the index of the ℓ-th occurrence of c in F. By Lemma 3, line 7 stores correctly in psi$[h]$ the value $i = \Psi(h)$. After the computation of array psi, s is retrieved by using the procedure psi2text of Fig. 2, whose correctness immediately follows from Theorem 5.

Clearly, the procedures bwt2psi and psi2text in Fig. 2 run in $O(n)$ time. Their working space is dominated by the cost of storing the array psi which takes $O(n \log n)$ bits. □

The Burrows–Wheeler Compression Algorithm

The rationale for using the bwt for data compression is the following. Consider a string w that appears k times within s. In the bwt matrix of s there will be k consecutive rows prefixed by w, say rows $r_w + 1, r_w + 2, \dots, r_w + k$. Hence, the positions $r_w + 1, \dots, r_w + k$ of $\hat{s} = \mathsf{bwt}(s)$ will contain precisely the symbols that immediately precede w in s. If in s certain patterns are more frequent than others, then for many substrings w the corresponding positions $r_w + 1, \dots, r_w + k$ of $\hat{s}$ will contain only a few distinct symbols. For example, if s is an English text and w is the string `his`, the corresponding portion of $\hat{s}$ will likely contain many `t`'s and blanks and only a few other symbols. Hence $\hat{s}$ is a permutation of s that is usually *locally homogeneous*, in that its "short" substrings usually contain only a few distinct symbols.[2]

To take advantage of this property, Burrows and Wheeler proposed to process the string $\hat{s}$ using move-to-front encoding [2] (mtf). mtf encodes each symbol with the number of distinct symbols encountered since its previous occurrence. To this end, mtf maintains a list of the symbols ordered by recency of occurrence; when the next symbol arrives the encoder outputs its current rank and moves it to the front of the list. Note that mtf produces

[2]Obviously this is true only if s has some regularity: if s is a random string $\hat{s}$ will be random as well!

a string which has the same length as $\hat{s}$ and, if $\hat{s}$ is locally homogeneous, the string mtf($\hat{s}$) will mainly consist of small integers.[3] Given this skewed distribution, mtf($\hat{s}$) can be easily compressed: Burrows and Wheeler proposed to compress it using Huffman or Arithmetic coding, possibly preceded by the run-length encoding of runs of equal integers.

Burrows and Wheeler were mainly interested in proposing an algorithm with good practical performance. Indeed their simple implementation outperformed, in terms of compression ratio, the tool gzip that was the current standard for lossless compression. A few years after the introduction of the bwt, [9,12] have shown that the compression ratio of the Burrows–Wheeler compression algorithm can be bounded in terms of the k-th order empirical entropy of the input string for any $k \geq 0$. For example, Kaplan et al. [9] showed that for any input string s and real $\mu > 1$, the length of the compressed string is bounded by $\mu n H_k(s) + n \log(\zeta(\mu)) + \mu g_k + O(\log n)$ bits, where $\zeta(\mu)$ is the standard Zeta function and g_k is a function depending only on k and the size of Σ. This bound holds *pointwise* for *any* string s, *simultaneously* for any $k \geq 0$ and $\mu > 1$, and it is remarkable since similar bounds have not been proven for any other known compressor. The theoretical study on the performance of bwt-based compressors is currently a very active area of research. The reader is referred to the recommended readings for further information.

Applications

After the seminal paper of Burrows and Wheeler, many researchers have proposed compression algorithms based on the bwt (see [4,5] and references therein). Of particular theoretical interest are the results in [6] showing that the bwt can be used to design a "compression booster", that is, a tool for improving the performance of other compressors in a well-defined and measurable way.

Another important area of application of the bwt is the design of Compressed Full-text Indexes [14]. These indexes take advantage of the relationship between the bwt and the Suffix Array to provide a compressed representation of a string supporting the efficient search and retrieval of the occurrences of an arbitrary pattern.

Open Problems

In addition to the investigation on the performance of bwt-based compressors, an open problem of great practical significance is the space efficient computation of the bwt. Given a string s of length n over an alphabet Σ, both s and $\hat{s}$ = bwt(s) take $O(n \log |\Sigma|)$ bits. Unfortunately, the linear time algorithms shown in Fig. 2 make use of auxiliary arrays (i. e. sa and psi) whose storage takes $\Theta(n \log n)$ bits. This poses a serious limitation to the size of the largest bwt that can be computed in main memory. Space efficient algorithms for inverting the bwt have been obtained in the compressed indexing literature [14], while the problem of space- and time-efficient computation of the bwt is still open even if interesting preliminary results are reported in [8,10,13].

[3] If s is an English text, mtf($\hat{s}$) usually contains more that 50% zeroes.

Experimental Results

An experimental study of the performance of several compression algorithms based on the bwt and a comparison with other state-of-the-art compressors is presented in [4].

Data Sets

The data sets used in [4] are available from http://www.mfn.unipmn.it/~manzini/boosting. Other data sets relevant for compression and compressed indexing are available at the Pizza&Chili site http://pizzachili.di.unipi.it/.

URL to Code

The Compression Boosting page (http://www.mfn.unipmn.it/~manzini/boosting) contains the source code of the algorithms tested in [4]. A more "lightweight" code for the computation of the bwt and its inverse (without compression) is available at http://www.mfn.unipmn.it/~manzini/lightweight. The code of bzip2 is available at http://www.bzip.org.

Cross References

- ▶ Arithmetic Coding for Data Compression
- ▶ Boosting Textual Compression
- ▶ Compressed Suffix Array
- ▶ Compressed Text Indexing
- ▶ Suffix Array Construction
- ▶ Table Compression
- ▶ Tree Compression and Indexing

Recommended Reading

1. Bell, T.C., Cleary, J.G., Witten, I.H.: Text compression. Prentice Hall, NJ (1990)
2. Bentley, J., Sleator, D., Tarjan, R., Wei, V.: A locally adaptive compression scheme. Commun. ACM **29**, 320–330 (1986)

3. Burrows, M., Wheeler, D.: A block sorting lossless data compression algorithm. Tech. Report 124, Digital Equipment Corporation (1994)
4. Ferragina, P., Giancarlo, R., Manzini, G.: The engineering of a compression boosting library: Theory vs practice in bwt compression. In: Proc. 14th European Symposium on Algorithms (ESA). LNCS, vol. 4168, pp. 756–767. Springer, Berlin (2006)
5. Ferragina, P., Giancarlo, R., Manzini, G.: The myriad virtues of wavelet trees. In: Proc. 33th International Colloquium on Automata and Languages (ICALP), pp. 561–572. LNCS n. 4051. Springer, Berlin, Heidelberg (2006)
6. Ferragina, P., Giancarlo, R., Manzini, G., Sciortino, M.: Boosting textual compression in optimal linear time. J. ACM **52**, 688–713 (2005)
7. Ferragina, P., Luccio, F., Manzini, G., Muthukrishnan, S.: Structuring labeled trees for optimal succinctness, and beyond. In: Proc. 46th IEEE Symposium on Foundations of Computer Science (FOCS), 184–193, Pittsburgh, PA (2005)
8. Hon, W., Sadakane, K., Sung, W.: Breaking a time-and-space barrier in constructing full-text indices. In: Proc. of the 44th IEEE Symposium on Foundations of Computer Science (FOCS), 251–260, Cambridge, MA (2003)
9. Kaplan, H., Landau, S., Verbin, E.: A simpler analysis of Burrows-Wheeler-based compression. Theoretical Computer Science **387**(3): 220–235 (2007)
10. Kärkkäinen, J.: Fast BWT in small space by blockwise suffix sorting. Theoretical Computer Science **387**(3): 249–257 (2007)
11. Kärkkäinen, J., Sanders, P., Burkhardt, S.: Linear work suffix array construction. J. ACM **53**(6), 918–936 (2006)
12. Manzini, G.: An analysis of the Burrows-Wheeler transform. J. ACM **48**, 407–430 (2001)
13. Na, J.: Linear-time construction of compressed suffix arrays using $o(n \log n)$-bit working space for large alphabets. In: Proc. 16th Symposium on Combinatorial Pattern Matching (CPM). LNCS, vol. 3537, pp. 57–67. Springer, Berlin (2005)
14. Navarro, G., Mäkinen, V.: Compressed full text indexes. ACM Comput. Surv. **39**(1) (2007)
15. Salomon, D.: Data Compression: the Complete Reference, 3rd edn. Springer, New York (2004)
16. Vo, B.D., Vo, K.P.: Using column dependency to compress tables. In: Proc. of IEEE Data Compression Conference (DCC), pp. 92–101, IEEE Computer Society Press (2004)

Byzantine Agreement

1980; Pease, Shostak, Lamport

MICHAEL OKUN
Weizmann Institute of Science, Rehovot, Israel

Keywords and Synonyms

Consensus; Byzantine generals; Interactive consistency

Problem Definition

The study of Pease, Shostak and Lamport was among the first to consider the problem of achieving a coordinated behavior between processors of a distributed system in the presence of failures [21]. Since the paper was published, this subject has grown into an extensive research area. Below is a presentation of the main findings regarding the specific questions addressed in their paper. In some cases this entry uses the currently accepted terminology in this subject, rather than the original terminology used by the authors.

System Model

A distributed system is considered to have n independent processors, $p_1, \ldots, p_n$, each modeled as a (possibly infinite) state machine. The processors are linked by a communication network that supports direct communication between every pair of processors. The processors can communicate only by exchanging messages, where the sender of every message can be identified by the receiver. While the processors may fail, it is assumed that the communication subsystem is fail-safe. It is not known in advance which processors will not fail (remain correct) and which ones will fail. The types of processor failures are classified according to the following hierarchy.

Crash failure A crash failure means that the processor no longer operates (ad infinitum, starting from the failure point). In particular, other processors will not receive messages from a faulty processor after it crashes.
Omission failure A processor fails to send and receive an arbitrary subset of its messages.
Byzantine failure A faulty processor behaves arbitrarily.

The Byzantine failure is further subdivided into two cases, according to the ability of the processors to create unfalsifiable signatures for their messages. In the *authenticated Byzantine failure* model it is assumed that each message is *signed* by its sender and that no other processor can fake a signature of a correct processor. Thus, even if such a message is forwarded by other processors, its authenticity can be verified. If the processors represent malevolent (human) users of a distributed system, a Public Key Infrastructure (PKI) is typically used to sign the messages (which involves cryptography related issues [17], not discussed here). Practically, in systems where processors are just "processors", a simple signature, such as CRC (cyclic redundancy check), might be sufficient [13]. In the *unauthenticated Byzantine failure* model there are no message signatures.

Definition of the Byzantine Agreement Problem

In the beginning, each processor p_i has an externally provided input value v_i, from some set V (of at least size 2). In the *Byzantine Agreement* (BA) problem, every correct processor p_i is required to decide on an output value $d_i \in V$ such that the following conditions hold:

Termination Eventually, p_i decides, i. e., the algorithm cannot run indefinitely.

Validity If the input value of all the processors is v, then the correct processors decide v.

Agreement All the correct processors decide on the same value.

For crash failures and omission failures there exists a stronger agreement condition:

Uniform Agreement No two processors (either correct or faulty) decide differently.

The termination condition has the following stronger version.

Simultaneous Termination All the correct processors decide in the *same round* (see definition below).

Timing Model

The BA problem was originally defined for *synchronous* distributed systems [18,21]. In this timing model the processors are assumed to operate in lockstep, which allows to partition the execution of a protocol to rounds. Each round consists of a send phase, during which a processor can send a (different) message to each processor directly connected to it, followed by a receive phase, in which it receives messages sent by these processors in the current round. Unlimited local computations (state transitions) are allowed in both phases, which models the typical situation in real distributed systems, where computation steps are faster than the communication steps by several orders of magnitude.

Overview

This entry deals only with *deterministic* algorithms for the BA problem in the synchronous model. For algorithms involving randomization see the ► Probabilistic Synchronous Byzantine Agreement entry in this volume. For results on BA in other models of synchrony, see ► Asynchronous Consensus Impossibility, ► Failure Detectors, ► Consensus with Partial Synchrony entries in this volume.

Key Results

The maximum possible number of faulty processors is assumed to be bounded by an a priori specified number t (e. g., estimated from the failure probability of individual processor and the requirements on the failure probability of the system as a whole). The number of processors that actually become faulty in a given execution is denoted by f, where $f \leq$ t.

The complexity of synchronous distributed algorithms is measured by three complementary parameters. The first is the *round complexity,* which measures the number of rounds required by the algorithm. The second is the *message complexity*, i. e., the total number of messages (and sometimes also their size in bits) sent by all the processors (in case of Byzantine failures, only messages sent by correct processors are counted). The third complexity parameter measures the number of local operations, as in sequential algorithms.

All the algorithms presented bellow are efficient, i. e., the number of rounds, the number of messages and their size, and the local operations performed by each processor are polynomial in n. In most of the algorithms, both the exchanged messages and the local computations involve only the basic data structures (e. g., arrays, lists, queues). Thus, the discussion is restricted only to the round and the message complexities of the algorithms.

The network is assumed to be fully connected, unless explicitly stated otherwise.

Crash Failures

A simple BA algorithm which runs in $t + 1$ rounds and sends $O(n^2)$ messages, together with a proof that this number of rounds is optimal, can be found in textbooks on distributed computing [19]. Algorithms for deciding in $f + 1$ rounds, which is the best possible, are presented in [7,23] (one additional round is necessary before the processors can stop [11]). Simultaneous termination requires $t + 1$ rounds, even if no failures actually occur [11], however there exists an algorithm that in any given execution stops in the earliest possible round [14]. For uniform agreement, decision can be made in $\min(f + 2, t + 1)$ rounds, which is tight [7].

In case of crash failures it is possible to solve the BA problem with $O(n)$ messages, which is also the lower bound. However, all known message-optimal BA algorithms require a superlinear time. An algorithm that runs in $O(f + 1)$ rounds and uses only $O(n \text{ polylog } n)$ messages, is presented in [8], along with an overview of other results on BA message complexity.

Omission Failures

The basic algorithm used to solve the crash failure BA problem works for omission failures as well, which allows to solve the problem in $t + 1$ rounds [23]. An algorithm which terminates in $\min(f + 2, t + 1)$ rounds was presented in [22]. Uniform agreement is impossible for $t \geq n/2$ [23]. For $t < n/2$, there is an algorithm that achieves uniform agreement in $\min(f + 2, t + 1)$ rounds (and $O(n^2 f)$ message complexity) [20].

Byzantine Failures with Authentication

A $(t + 1)$-round BA algorithm is presented in [12]. An algorithm which terminates in $\min(f + 2, t + 1)$ rounds can be found in [24]. The message complexity of the problem is analyzed in [10], where it is shown that the number of signatures and the number of messages in any authenticated BA algorithm are $\Omega(nt)$ and $\Omega(n + t^2)$, respectively. In addition, it is shown that $\Omega(nt)$ is the bound on the number of messages for the unauthenticated BA.

Byzantine Failures Without Authentication

In the unauthenticated case, the BA problem can be solved if and only if $n > 3t$. The proof can be found in [1,19]. An algorithm that decides in $\min(f + 3, t + 1)$ rounds (it might require two additional rounds to stop) is presented in [16]. Unfortunately, this algorithm is complicated. Simpler algorithms, that run in $\min(2f + 4, 2t + 1)$ and $3\min(f + 2, t + 1)$ rounds, are presented in [24] and [5], respectively. In these algorithms the number of sent messages is $O(n^3)$, moreover, in the latter algorithm the messages are of constant size (2 bits). Both algorithms assume $V = \{0, 1\}$. To solve the BA problem for a larger V, several instances of a binary algorithm can be run in parallel. Alternatively, there exists a simple 2-round protocol that reduces a BA problem with arbitrary initial values to the binary case, e. g., see Sect. 6.3.3 in [19]. For algorithms with optimal $O(nt)$ message complexity and $t + o(t)$ round complexity see [4,9].

Arbitrary Network Topologies

When the network is not fully connected, BA can be solved for crash, omission and authenticated Byzantine failures if and only if it is $(t + 1)$-connected [12]. In case of Byzantine failures without authentication, BA has a solution if and only if the network is $(2t + 1)$-connected and $n > 3t$ [19]. In both cases the BA problem can be solved by simulating the algorithms for the fully connected network, using the fact that the number of disjoint communication paths between any pair of non-adjacent processors exceeds the number of faulty nodes by an amount that is sufficient for reliable communication.

Interactive Consistency and Byzantine Generals

The BA (consensus) problem can be stated in several similar ways. Two widely used variants are the *Byzantine Generals* (BG) problem and the *Interactive Consistency* (IC) problem. In the BG case there is a designated processor, say p_1, which is the only one to have an input value. The termination and agreement requirements of the BG problem are exactly as in BA, while the validity condition requires that if the input value of p_1 is v and p_1 is correct, then the correct processors decide v. The IC problem is an extension of BG, where every processor is "designated", so that each processor has to decide on a vector of n values, where the conditions for the i-th entry are as in BG, with p_i as the designated processor. For deterministic synchronous algorithms BA, BG and IC problems are essentially equivalent, e. g., see the discussion in [15].

Firing Squad

The above algorithms assume that the processors share a "global time", i. e., all the processors start in the same (first) round, so that their round counters are equal throughout the execution of the algorithm. However, there are cases in which the processors run in a synchronous network, yet each processor has its own notion of time (e. g., when each processor starts on its own, the round counter values are distinct among the processors). In these cases, it is desirable to have a protocol that allows the processors to agree on some specific round, thus creating a common round which synchronizes all the correct processors. This synchronization task, known as the *Byzantine firing squad* problem [6], is tightly realted to BA.

General Translation Techniques

One particular direction that was pursued as part of the research on the BA problem is the development of methods that automatically translate any protocol that tolerates a more benign failure type into one which tolerates more severe failures [24]. Efficient translations spanning the entire failure hierarchy, starting from crash failures all the way to unauthenticated Byzantine failures, can be found in [3] and in Ch. 12 of [1].

Applications

Due to the very tight synchronization assumptions made in the algorithms presented above, they are used mainly in real-time, safety-critical systems, e. g., aircraft control [13]. In fact, the original interest of Pease, Shostak and Lamport in this problem was raised by such an application [21]. In addition, BA protocols for the Byzantine failure case serve as a basic building block in many cryptographic protocols, e. g., secure multi-party computation [17], by providing a broadcast channel on top of pairwise communication channels.

Cross References

- ▶ Asynchronous Consensus Impossibility
- ▶ Atomic Broadcast
- ▶ Consensus with Partial Synchrony
- ▶ Failure Detectors
- ▶ Optimal Probabilistic Synchronous Byzantine Agreement
- ▶ Randomization in Distributed Computing
- ▶ Renaming
- ▶ Set Agreement

Recommended Reading

1. Attiya, H., Welch, J.L.: Distributed Computing: Fundamentals, Simulations and Advanced Topics. McGraw-Hill, UK (1998)
2. Barborak, M., Dahbura, A., Malek, M.: The Consensus Problem in Fault-Tolerant Computing. ACM Comput. Surv. **25**(2), 171–220 (1993)
3. Bazzi, R.A., Neiger, G.: Simplifying Fault-tolerance: Providing the Abstraction of Crash Failures. J. ACM **48**(3), 499–554 (2001)
4. Berman, P., Garay, J.A., Perry, K.J.: Bit Optimal Distributed Consensus. In: Yaeza-Bates, R., Manber, U. (eds.) Computer Science Research, pp. 313–322. Plenum Publishing Corporation, New York (1992)
5. Berman, P., Garay, J.A., Perry, K.J.: Optimal Early Stopping in Distributed Consensus. In: Proc. 6th International Workshop on Distributed Algorithms (WDAG), pp. 221–237, Israel, November 1992
6. Burns, J.E., Lynch, N.A.: The Byzantine Firing Squad problem. Adv. Comput. Res. **4**, 147–161 (1987)
7. Charron-Bost, B., Schiper, A.: Uniform Consensus is Harder than Consensus. J. Algorithms **51**(1), 15–37 (2004)
8. Chlebus, B.S., Kowalski, D.R.: Time and Communication Efficient Consensus for Crash Failures. In: Proc. 20th International Symposium on Distributed Computing (DISC), pp. 314–328, Sweden, September 2006
9. Coan, B.A., Welch, J.L.: Modular construction of a Byzantine agreement protocol with optimal message bit complexity. Inf. Comput. **97**(1), 61–85 (1992)
10. Dolev, D., Reischuk, R.: Bounds on Information Exchange for Byzantine Agreement. J. ACM **32**(1), 191–204 (1985)
11. Dolev, D., Reischuk, R., Strong, H.R.: Early Stopping in Byzantine Agreement. J. ACM **37**(4), 720–741 (1990)
12. Dolev, D., Strong, H.R.: Authenticated Algorithms for Byzantine Agreement. SIAM J. Comput. **12**(4), 656–666 (1983)
13. Driscoll, K., Hall, B., Sivencrona, H., Zumsteg, P.: Byzantine Fault Tolerance, from Theory to Reality. In: Proc. 22nd International Conference on Computer Safety, Reliability, and Security (SAFECOMP), pp. 235–248, UK, September 2003
14. Dwork, C., Moses, Y.: Knowledge and Common Knowledge in a Byzantine Environment: Crash Failures. Inf. Comput. **88**(2), 156–186 (1990)
15. Fischer, M.J.: The Consensus Problem in Unreliable Distributed Systems (A Brief Survey). Research Report, YALEU/DCS/RR-273, Yale University, New Heaven (1983)
16. Garay, J.A., Moses, Y.: Fully Polynomial Byzantine Agreement for $n > 3t$ Processors in $t + 1$ Rounds. SIAM J. Comput. **27**(1), 247–290 (1998)
17. Goldreich, O.: Foundations of Cryptography, vol. 1-2. Cambridge University Press, UK (2001) (2004)
18. Lamport, L., Shostak, R.E., Pease, M.C.: The Byzantine Generals Problem. ACM Trans. Program. Lang. Syst. **4**(3), 382–401 (1982)
19. Lynch, N.A.: Distributed Algorithms. Morgan Kaufmann, CA (1996)
20. Parvédy, P.R., Raynal, M.: Optimal Early Stopping Uniform Consensus in Synchronous Systems with Process Omission Failures. In: Proc. 16th Annual ACM Symposium on Parallel Algorithms (SPAA), pp. 302–310, Spain, June 2004
21. Pease, M.C., Shostak, R.E., Lamport, L.: Reaching Agreement in the Presence of Faults. J. ACM **27**(2), 228–234 (1980)
22. Perry, K.J., Toueg, S.: Distributed Agreement in the Presence of Processor and Communication Faults. IEEE Trans. Softw. Eng. **12**(3), 477–482 (1986)
23. Raynal, M.: Consensus in Synchronous Systems: A Concise Guided Tour. In: Proc. 9th Pacific Rim International Symposium on Dependable Computing (PRDC), pp. 221–228, Japan, December 2002
24. Toueg, S., Perry, K.J., Srikanth, T.K.: Fast Distributed Agreement. SIAM J. Comput. **16**(3), 445–457 (1987)

C

Cache-Oblivious B-Tree

2005; Bender, Demaine, Farach-Colton

ROLF FAGERBERG
Department of Mathematics and Computer Science,
University of Southern Denmark,
Odense, Denmark

Keywords and Synonyms

Cache-oblivious search tree; Cache-oblivious dictionary

Problem Definition

Computers contain a hierarchy of memory levels, with vastly differing access times. Hence, the time for a memory access depends strongly on what is the innermost level containing the data accessed. In analysis of algorithms, the standard RAM (or von Neumann) model cannot capture this effect, and external memory models were introduced to better model the situation. The most widely used of these models is the two-level I/O-model [1], also called the External Memory model or the Disk Access model. The I/O-model approximates the memory hierarchy by modeling two levels, with the inner level having size M, the outer level having infinite size, and transfers between the levels taking place in blocks of B consecutive elements. The cost of an algorithm is the number of memory transfers it makes.

The cache-oblivious model, introduced by Frigo et al. [18], elegantly generalizes the I/O-model to a multilevel memory model by a simple measure: the algorithm is not allowed to know the value of B and M. More precisely, a cache-oblivious algorithm is an algorithm formulated in the RAM model, but analyzed in the I/O-model, with an analysis valid for *any* value of B and M. Cache replacement is assumed to take place automatically by an optimal off-line cache replacement strategy. Since the analysis holds for any B and M, it holds for all levels simultaneously.

The subject here is that of efficient cache-oblivious data structures for the ordered dictionary problem, i. e., the problem of storing elements with keys from an ordered universe while supporting searches, insertions, deletions, and range searches. In full generality, searches are predecessor searches, returning the element with the largest key smaller than or equal to the key searched for.

Key Results

The first cache-oblivious dictionary was given by Prokop [21], who showed how to lay out a static binary tree in memory such that searches take $O(\log_B n)$ memory transfers. This layout, often called the *van Emde Boas layout* because it is reminiscent of the classic van Emde Boas data structure, also ensures that range searches take $O(\log_B n + k/B)$ memory transfers [2], where k is the size of the output. Both bounds are optimal for comparison-based searching.

The first dynamic, cache-oblivious dictionary was given by Bender et al. [10]. Making use of a variant of the van Emde Boas layout, a density maintenance algorithm of the type invented by Itai et al. [19], and weight-balanced B-trees [5], they arrived at the following results:

Theorem 1 ([10]) *There is a cache-oblivious dictionary structure supporting searches in $O(\log_B n)$ memory transfers, and insertions and deletions in amortized $O(\log_B n)$ memory transfers.*

Theorem 2 ([10]) *There is a cache-oblivious dictionary structure supporting searches in $O(\log_B n)$ memory transfers, insertions and deletions in amortized $O(\log_B n + (\log^2 n)/B)$ memory transfers, and range searches in $O(\log_B n + k/B)$ memory transfers, where k is the size of the output.*

Later, Bender et al. [7] developed a cache-oblivious structure for maintaining linked lists which supports insertion and deletion of elements in $O(1)$ memory transfers and scanning of k consecutive elements in amortized $O(k/B)$ memory transfers. Combining this structure with the structure of the first theorem above, the following result can be achieved.

Theorem 3 ([7,10]) *There is a cache-oblivious dictionary structure supporting searches in $O(\log_B n)$ memory transfers, insertions and deletions in amortized $O(\log_B n)$ memory transfers, and range searches in amortized $O(\log_B n + k/B)$ memory transfers, where k is the size of the output.*

A long list of extensions of these basic cache-oblivious dictionary results has been given. We now survey these.

Bender et al. [11] and Brodal et al. [16] gave very similar proposals for reproducing the result of Theorem 2, but with significantly simpler structures (avoiding the use of weight-balanced B-trees). On the basis of exponential trees, Bender et al.[8] gave a proposal with $O(\log_B n)$ worst-case queries and updates. They also gave a solution with partial persistence, where searches (in all versions of the structure) and updates (in the latest version of the structure) require amortized $O(\log_B(m + n))$ memory transfers, where m is the number of versions and n is the number of elements in the version operated on. Bender et al. [14] extended the cache-oblivious model to a concurrent setting, and gave three proposals for cache-oblivious B-trees in this setting. Bender et al. [12] gave cache-oblivious dictionary structures exploring trade-offs between faster insertion costs and slower search cost. Franceschini and Grossi [17] showed how to achieve $O(\log_B n)$ worst-case queries and updates while using $O(1)$ space besides the space for the n elements stored. Extensions to dictionaries on other data types such as strings [13,15] and geometric data [3,4,6] have been given.

It has been shown [9] that the best-possible multiplicative constant in the $\Theta(\log_B n)$ search bound for comparison-based searching is different in the I/O-model and in the cache-oblivious model.

Applications

Dictionaries solve a fundamental data structuring problem which is part of solutions for a very high number of computational problems. Dictionaries for external memory are useful in settings where memory accesses are dominating the running time, and cache-oblivious dictionaries in particular stand out by their ability to optimize the access to all levels of an unknown memory hierarchy. This is an asset e. g. when developing programs to be run on diverse or unknown architectures (such as software libraries or programs for heterogeneous distributed computing like grid computing and projects such as SETI@home). Even on a single, known architecture, the memory parameters available to a computational process may be non-constant if several processes compete for the same memory resources. Since cache-oblivious algorithms are optimized for all parameter values, they have the potential to adapt more gracefully to these changes, and also to varying input sizes forcing different memory levels to be in use.

Open Problems

For the one-dimensional ordered dictionary problem discussed here, one notable open problem is to find a data structure achieving worst case versions of all the bounds in Theorem 3.

Experimental Results

Cache-oblivious dictionaries have been evaluated empirically in [11,13,16,20,22]. The overall conclusion of these investigations is that cache-oblivious methods easily can outperform RAM algorithms, although sometimes not as much as algorithms tuned to the specific memory hierarchy and problem size in question. On the other hand, cache-oblivious algorithms seem to perform well on all levels of the memory hierarchy, and to be more robust to changing problem sizes.

Cross References

- ► B-trees
- ► Cache-Oblivious Model
- ► Cache-Oblivious Sorting
- ► I/O-model

Recommended Reading

1. Aggarwal, A., Vitter, J.S.: The Input/Output complexity of sorting and related problems. Commun. ACM **31**(9), 1116–1127 (1988)
2. Arge, L., Brodal, G.S., Fagerberg, R.: Cache-oblivious data structures. In: Mehta, D., Sahni, S. (eds.) Handbook on Data Structures and Applications. CRC Press, Boca Raton (2005)
3. Arge, L., Brodal, G.S., Fagerberg, R., Laustsen, M.: Cache-oblivious planar orthogonal range searching and counting. In: Proc. 21st ACM Symposium on Computational Geometry, pp. 160–169. ACM, New York (2005)
4. Arge, L., de Berg, M., Haverkort, H.J.: Cache-oblivious R-trees. In: Proc. 21st ACM Symposium on Computational Geometry, pp. 170–179. ACM, New York (2005)
5. Arge, L., Vitter, J.S.: Optimal external memory interval management. SIAM J. Comput. **32**(6), 1488–1508 (2003)
6. Arge, L., Zeh, N.: Simple and semi-dynamic structures for cache-oblivious planar orthogonal range searching. In: Proc. 22nd ACM Symposium on Computational Geometry, pp. 158–166. ACM, New York (2006)

7. Bender, M., Cole, R., Demaine, E., Farach-Colton, M.: Scanning and traversing: Maintaining data for traversals in a memory hierarchy. In: Proc. 10th Annual European Symposium on Algorithms. LNCS, vol. 2461, pp. 139–151. Springer, Berlin (2002)
8. Bender, M., Cole, R., Raman, R.: Exponential structures for cache-oblivious algorithms. In: Proc. 29th International Colloquium on Automata, Languages, and Programming. LNCS, vol. 2380, pp. 195–207. Springer, Berlin (2002)
9. Bender, M.A., Brodal, G.S., Fagerberg, R., Ge, D., He, S., Hu, H., Iacono, J., Lopez-Ortiz, A.: The cost of cache-oblivious searching. In: Proc. 44th Annual IEEE Symposium on Foundations of Computer Science, pp. 271–282. IEEE Computer Society Press, Los Alamitos (2003)
10. Bender, M.A., Demaine, E.D., Farach-Colton, M.: Cache-oblivious B-trees. SIAM J. Comput. **35**(2), 341–358 (2005). Conference version appeared at FOCS (2000)
11. Bender, M.A., Duan, Z., Iacono, J., Wu, J.: A locality-preserving cache-oblivious dynamic dictionary. J. Algorithms **53**(2), 115–136 (2004). Conference version appeared at SODA (2002)
12. Bender, M.A., Farach-Colton, M., Fineman, J.T., Fogel, Y.R., Kuszmaul, B.C., Nelson, J.: Cache-oblivious streaming B-trees. In: Proc. 19th Annual ACM Symposium on Parallel Algorithms and Architectures, pp. 81–92. ACM, New York (2007)
13. Bender, M.A., Farach-Colton, M., Kuszmaul, B.C.: Cache-oblivious string B-trees. In: Proc. 25th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, pp. 233–242. ACM, New York (2006)
14. Bender, M.A., Fineman, J.T., Gilbert, S., Kuszmaul, B.C.: Concurrent cache-oblivious B-trees. In: Proc. 17th Annual ACM Symposium on Parallel Algorithms, pp. 228–237. ACM, New York (2005)
15. Brodal, G.S., Fagerberg, R.: Cache-oblivious string dictionaries. In: SODA: ACM-SIAM Symposium on Discrete Algorithms, pp. 581–590. ACM Press, New York (2006)
16. Brodal, G.S., Fagerberg, R., Jacob, R.: Cache-oblivious search trees via binary trees of small height. In: Proc. 13th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 39–48 ACM, New York (2002)
17. Franceschini, G., Grossi, R.: Optimal worst-case operations for implicit cache-oblivious search trees. In: Proc. Algorithms and Data Structures, 8th International Workshop, WADS. LNCS, vol. 2748, pp. 114–126. Springer, Berlin (2003)
18. Frigo, M., Leiserson, C.E., Prokop, H., Ramachandran, S.: Cache-oblivious algorithms. In: 40th Annual IEEE Symposium on Foundations of Computer Science, pp. 285–298. IEEE Computer Society Press, Los Alamitos (1999)
19. Itai, A., Konheim, A.G., Rodeh, M.: A sparse table implementation of priority queues. In: Automata, Languages and Programming, 8th Colloquium. LNCS, vol. 115, pp. 417–431. Springer, Berlin (1981)
20. Ladner, R.E., Fortna, R., B.-Nguyen, H.: A comparison of cache aware and cache oblivious static search trees using program instrumentation. In: Experimental Algorithmics. LNCS, vol. 2547, pp. 78–92. Springer, Berlin (2000)
21. Prokop, H.: Cache-oblivious algorithms. Master's thesis, Massachusetts Institute of Technology (1999)
22. Rahman, N., Cole, R., Raman, R.: Optimised predecessor data structures for internal memory. In: Proc. Algorithm Engineering, 5th International Workshop, WAE. LNCS, vol. 2141, pp. 67–78. Springer, Berlin (2001)

Cache-Oblivious Model

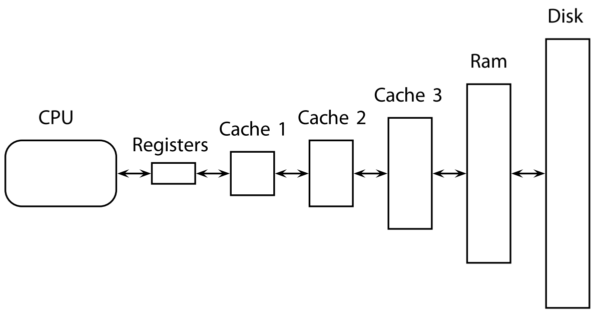
1999; Frigo, Leiserson, Prokop, Ramachandran

Rolf Fagerberg
Department of Mathematics and Computer Science, University of Southern Denmark, Odense, Denmark

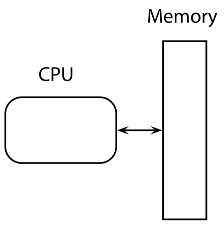
Model Definition

The memory system of contemporary computers consists of a hierarchy of memory levels, with each level acting as a cache for the next; a typical hierarchy may consist of registers, level 1 cache, level 2 cache, level 3 cache, main memory, and disk (Fig. 1). One characteristic of the hierarchy is that the memory levels get larger and slower the further they get from the processor, with the access time increasing most dramatically between RAM memory and disk. Another characteristic is that data is moved between levels in blocks.

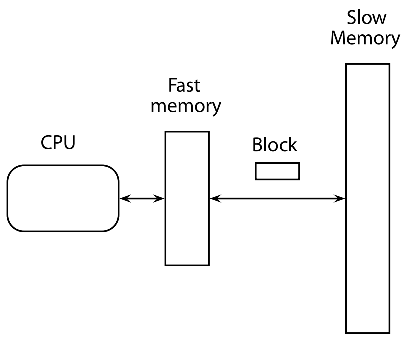
As a consequence of the differences in access time between the levels, the cost of a memory access depends highly on what is the current lowest memory level holding the element accessed. Hence, the memory access pattern of an algorithm has a major influence on its practical running time. Unfortunately, the RAM model (Fig. 2) traditionally used to design and analyze algorithms is not



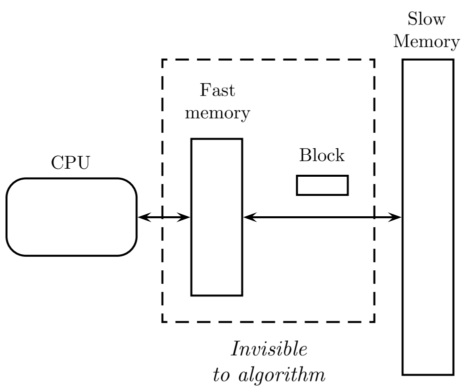
Cache-Oblivious Model, Figure 1
The memory hierarchy



Cache-Oblivious Model, Figure 2
The RAM model



Cache-Oblivious Model, Figure 3
The I/O-model



Cache-Oblivious Model, Figure 4
The cache-oblivious model

capable of capturing this, as it assumes that all memory accesses take equal time.

To better account for the effects of the memory hierarchy, a number of computational models have been proposed. The simplest and most successful is the two-level I/O-model introduced by Aggarwal and Vitter [2] (Fig. 3). In this model a two-level memory hierarchy is assumed, consisting of a fast memory of size M and a slower memory of infinite size, with data transferred between the levels in blocks of B consecutive elements. Computation can only be performed on data in the fast memory, and algorithms are assumed to have complete control over transfers of blocks between the two levels. Such a block transfer is denoted a *memory transfer*. The complexity measure is the number of memory transfers performed. The strength of the I/O-model is that it captures part of the memory hierarchy, while being sufficiently simple to make design and analysis of algorithms feasible. Over the last two decades, a large body of results for the I/O-model has been produced, covering most areas of algorithmics. For an overview, see the surveys [3,24,26,27].

More elaborate models of multi-level memory have been proposed (see e. g. [26] for an overview) but these models have been less successful than the I/O-model, mainly because of their complexity which makes analysis of algorithms harder. All these models, including the I/O-model, assume that the characteristics of the memory hierarchy (the level and block sizes) are known.

In 1999 the *cache-oblivious model* (Fig. 4) was introduced by Frigo et al. [22]. A cache-oblivious algorithm is an algorithm formulated in the RAM model but analyzed in the I/O-model, with the analysis required to hold for *any* block size B and memory size M. Memory transfers are assumed to take place automatically by an optimal off-line cache replacement strategy.

The crux of the cache-oblivious model is that because the I/O-model analysis holds for any block and memory size, it holds for all levels of a multi-level memory hierarchy (see [22,25] for detailed versions of this statement). Put differently, by optimizing an algorithm to one unknown level of the memory hierarchy, it is optimized to all levels simultaneously. Thus, the cache-oblivious model elegantly generalizes the I/O-model to a multi-level memory model by one simple measure: the algorithm is not allowed to know the value of B and M. The challenge, of course, is to develop algorithms having good memory transfer analyzes under these conditions.

Besides capturing the entire memory hierarchy in a conceptually simple way, the cache-oblivious model has other benefits: Algorithms developed in the model do not rely on knowing the parameters of the memory hierarchy, which is an asset when developing programs to be run on diverse or unknown architectures (e. g. software libraries or programs for heterogeneous distributed computing such as grid computing and projects like SETI@home). Even on a single, known architecture, the memory parameters available to a computational process may be nonconstant if several processes compete for the same memory resources. Since cache-oblivious algorithms are optimized for all parameter values, they have the potential to adapt more gracefully to these changes. Also, the same code will adapt to varying input sizes forcing different memory levels to be in use. Finally, cache-oblivious algorithms automatically are optimizing the use of translation lookaside buffers (a cache holding recently accessed parts of the page table used for virtual memory) of the CPU, which may be seen as a second memory hierarchy parallel to the one mentioned in the introduction.

Possible weak points of the cache-oblivious model are the assumption of optimal off-line cache replacement, and the lack of modeling of the limited associativity of many of the levels of the hierarchy. The first point is mitigated by the fact that normally, the provided analysis of a proposed cache-oblivious algorithm will work just as well assuming a Least-Recently-Used cache replacement policy, which is closer to actual replacement strategies of computers. The second point is also a weak point of most other memory models.

Key Results

This section surveys a number of the known results in the cache-oblivious model. Other surveys available include [5,14,20,24].

First of all, note that scanning an array of N elements takes $O(N/B)$ memory transfers for any values of B and M, and hence is an optimal cache-oblivious algorithm. Thus, standard RAM algorithms based on scanning may already possess good analysis in the cache-oblivious model – for instance, the classic deterministic selection algorithm has complexity $O(N/B)$ [20].

For sorting, a fundamental fact in the I/O-model is that comparison-based sorting of N elements takes $\Theta(\mathrm{Sort}(N))$ memory transfers [2], where $\mathrm{Sort}(N) = \frac{N}{B}\log_{M/B}\frac{N}{M}$. Also in the cache-oblivious model, sorting can be carried out in $\Theta(\mathrm{Sort}(N))$ memory transfer, if one makes the so-called *tall cache* assumption $M \geq B^{1+\varepsilon}$ [15,22]. Such an assumption has been shown to be necessary [16], which proves a separation in power between cache-oblivious algorithms and algorithms in the I/O-model (where this assumption is not needed for the sorting bound).

For searching, B-trees have cost $O(\log_B N)$, which is optimal in the I/O-model for comparison-based searching. This cost is also attainable in the cache-oblivious model, as shown for the static case in [25] and for the dynamic case in [13]. A number of later variants of cache-oblivious search trees have appeared. Also for searching, a separation between cache-oblivious algorithms and algorithms in the I/O-model has been shown [12] in the sense that the constants attainable in the $O(\log_B N)$ bound are provably different.

Permuting in the I/O-model has complexity $\Theta(\min\{\mathrm{Sort}(N), N\})$, assuming that elements are indivisible [2]. It has been proven [16] that this asymptotic complexity cannot be attained in the cache-oblivious model, hence also for this problem, a separation exists.

Cache-oblivious priority queues supporting operations in $O(1/B \log_{M/B} N/M)$ memory transfers amortized have been given.

Currently known cache-oblivious algorithms also include algorithms for problems in computational geometry [1,6,7,8,10,15], for graph problems [4,17,18,23], for scanning dynamic sets [9], for layout of static trees [11], for search problems on multi-sets [21], for dynamic programming [19], for partial persistence [10], for matrix operations [22], and for the Fast Fourier Transform [22].

Applications

The cache-oblivious model is a means for design and analysis of algorithms that use the memory hierarchy of computers efficiently.

Experimental Results

Cache-oblivious algorithms have been evaluated empirically in a number of areas, including sorting, searching, matrix algorithms [22], and dynamic programming [19]. The overall conclusion of these investigations is that cache-oblivious methods often outperform RAM algorithms, but not always exactly as much as do algorithms tuned to the specific memory hierarchy and problem size. On the other hand, cache-oblivious algorithms seem to perform well on all levels of the memory hierarchy, and to be more robust to changing problem sizes.

Cross References

► Cache-Oblivious B-Tree
► Cache-Oblivious Sorting
► I/O-model

Recommended Reading

1. Agarwal, P.K., Arge, L., Danner, A., Holland-Minkley, B.: Cache-oblivious data structures for orthogonal range searching. In: Proc. 19th ACM Symposium on Computational Geometry, pp. 237–245. ACM, New York (2003)
2. Aggarwal, A., Vitter, J.S.: The Input/Output complexity of sorting and related problems. Commun. ACM **31**(9), 1116–1127 (1988)
3. Arge, L.: External memory data structures. In: Abello, J., Pardalos, P.M., Resende, M.G.C. (eds.) Handbook of Massive Data Sets, pp. 313–358. Kluwer Academic Publishers, Boston (2002)
4. Arge, L., Bender, M.A., Demaine, E.D., Holland-Minkley, B., Munro, J.I.: Cache-oblivious priority queue and graph algorithm applications. In: Proc. 34th Annual ACM Symposium on Theory of Computing, pp. 268–276. ACM, New York (2002)
5. Arge, L., Brodal, G.S., Fagerberg, R.: Cache-oblivious data structures. In: Mehta, D., Sahni, S. (eds.) Handbook on Data Structures and Applications. CRC Press, Boca Raton (2005)
6. Arge, L., Brodal, G.S., Fagerberg, R., Laustsen, M.: Cache-oblivious planar orthogonal range searching and counting. In: Proc. 21st Annual ACM Symposium on Computational Geometry, pp. 160–169. ACM, New York (2005)
7. Arge, L., de Berg, M., Haverkort, H.J.: Cache-oblivious R-trees. In: Symposium on Computational Geometry, pp. 170–179. ACM, New York (2005)

8. Arge, L., Zeh, N.: Simple and semi-dynamic structures for cache-oblivious planar orthogonal range searching. In: Symposium on Computational Geometry, pp. 158–166. ACM, New York (2006)
9. Bender, M., Cole, R., Demaine, E., Farach-Colton, M.: Scanning and traversing: Maintaining data for traversals in a memory hierarchy. In: Proc. 10th Annual European Symposium on Algorithms. LNCS, vol. 2461, pp. 139–151. Springer, Berlin (2002)
10. Bender, M., Cole, R., Raman, R.: Exponential structures for cache-oblivious algorithms. In: Proc. 29th International Colloquium on Automata, Languages, and Programming. LNCS, vol. 2380, pp. 195–207. Springer, Berlin (2002)
11. Bender, M., Demaine, E., Farach-Colton, M.: Efficient tree layout in a multilevel memory hierarchy. In: Proc. 10th Annual European Symposium on Algorithms. LNCS, vol. 2461, pp. 165–173. Springer, Berlin (2002). Full version at http://arxiv.org/abs/cs/0211010
12. Bender, M.A., Brodal, G.S., Fagerberg, R., Ge, D., He, S., Hu, H., Iacono, J., López-Ortiz, A.: The cost of cache-oblivious searching. In: Proc. 44th Annual IEEE Symposium on Foundations of Computer Science, pp. 271–282. IEEE Computer Society Press, Los Alamitos (2003)
13. Bender, M.A., Demaine, E.D., Farach-Colton, M.: Cache-oblivious *B*-trees. In: 41st Annual Symposium on Foundations of Computer Science, pp. 399–409. IEEE Computer Society Press, Los Alamitos (2000)
14. Brodal, G.S.: Cache-oblivious algorithms and data structures. In: Proc. 9th Scandinavian Workshop on Algorithm Theory. LNCS, vol. 3111, pp. 3–13. Springer, Berlin (2004)
15. Brodal, G.S., Fagerberg, R.: Cache oblivious distribution sweeping. In: Proc. 29th International Colloquium on Automata, Languages, and Programming. LNCS, vol. 2380, pp. 426–438. Springer, Berlin (2002)
16. Brodal, G.S., Fagerberg, R.: On the limits of cache-obliviousness. In: Proc. 35th Annual ACM Symposium on Theory of Computing, pp. 307–315. ACM, New York (2003)
17. Brodal, G.S., Fagerberg, R., Meyer, U., Zeh, N.: Cache-oblivious data structures and algorithms for undirected breadth-first search and shortest paths. In: Proc. 9th Scandinavian Workshop on Algorithm Theory. LNCS, vol. 3111, pp. 480–492. Springer, Berlin (2004)
18. Chowdhury, R.A., Ramachandran, V.: Cache-oblivious shortest paths in graphs using buffer heap. In: Proc. 16th Annual ACM Symposium on Parallelism in Algorithms and Architectures. ACM, New York (2004)
19. Chowdhury, R.A., Ramachandran, V.: Cache-oblivious dynamic programming. In: Proc. 17th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 591–600. ACM-SIAM, New York (2006)
20. Demaine, E.D.: Cache-oblivious algorithms and data structures. In: Proc. EFF summer school on massive data sets, LNCS. Springer, Berlin. To appear. Online version at http://theory.csail.mit.edu/edemaine/papers/BRICS2002/
21. Farzan, A., Ferragina, P., Franceschini, G., Munro, J.I.: Cache-oblivious comparison-based algorithms on multisets. In: Proc. 13th Annual European Symposium on Algorithms. LNCS, vol. 3669, pp. 305–316. Springer, Berlin (2005)
22. Frigo, M., Leiserson, C.E., Prokop, H., Ramachandran, S.: Cache oblivious algorithms. In: 40th Annual IEEE Symposium on Foundations of Computer Science, pp. 285–298. IEEE Computer Society Press, Los Alamitos (1999)
23. Jampala, H., Zeh, N.: Cache-oblivious planar shortest paths. In: Proc. 32nd International Colloquium on Automata, Languages, and Programming. LNCS, vol. 3580, pp. 563–575. Springer, Berlin (2005)
24. Meyer, U., Sanders, P., Sibeyn, J.F. (eds.): Algorithms for Memory Hierarchies. LNCS, vol. 2625. Springer, Berlin (2003)
25. Prokop, H.: Cache-oblivious algorithms. Master's thesis, Massachusetts Institute of Technology, Dept. of Electrical Engineering and Computer Science (1999)
26. Vitter, J.S.: External memory algorithms and data structures: Dealing with MASSIVE data. ACM Comput. Surv. **33**(2), 209–271 (2001)
27. Vitter, J.S.: Geometric and spatial data structures in external memory. In: Mehta, D., Sahni, S. (eds.) Handbook on Data Structures and Applications. CRC Press, Boca Raton (2005)

Cache-Oblivious Sorting

1999; Frigo, Leiserson, Prokop, Ramachandran

Gerth Stølting Brodal
Department of Computer Science, University of Aarhus, Århus, Denmark

Keywords and Synonyms

Funnel sort

Problem Definition

Sorting a set of elements is one of the most well-studied computational problems. In the cache-oblivious setting the first study of sorting was presented in 1999 in the seminal paper by Frigo et al. [8] that introduced the cache-oblivious framework for developing algorithms aimed at machines with (unknown) hierarchical memory.

Model

In the cache-oblivious setting the computational model is a machine with two levels of memory: a cache of limited capacity and a secondary memory of infinite capacity. The capacity of the cache is assumed to be M elements and data is moved between the two levels of memory in blocks of B consecutive elements. Computations can only be performed on elements stored in cache, i. e. elements from secondary memory need to be moved to the cache before operations can access the elements. Programs are written as acting directly on one unbounded memory, i. e. programs are like standard RAM programs. The necessary block transfers between cache and secondary memory are handled automatically by the model, assuming an optimal offline cache replacement strategy. The core assumption of the cache-oblivious model is that M and B are *unknown to* *the algorithm* whereas in the related I/O model introduced by Aggarwal and Vitter [1] the algorithms know M and B and the algorithms perform the block transfers explicitly. A thorough discussion of the cache-oblivious model and its relation to multi-level memory hierarchies is given in [8].

Sorting

For the sorting problem the input is an array of N elements residing in secondary memory, and the output is required to be an array in secondary memory storing the input elements in sorted order.

Key Results

In the I/O model tight upper and lower bounds were proved for the sorting problem and the problem of permuting an array [1]. In particular it was proved that sorting requires $\Theta(\frac{N}{B}\log_{M/B}\frac{N}{B})$ block transfers and permuting an array requires $\Theta(\min\{N, \frac{N}{B}\log_{M/B}\frac{N}{B}\})$ block transfers. Since lower bounds for the I/O model also hold for the cache-oblivious model, the lower bounds from [1] immediately give a lower bound of $\Omega(\frac{N}{B}\log_{M/B}\frac{N}{B})$ block transfers for cache-oblivious sorting and $\Omega(\min\{N, \frac{N}{B}\log_{M/B}\frac{N}{B}\})$ block transfers for cache-oblivious permuting. The upper bounds from [1] can not be applied to the cache-oblivious setting since these algorithms make explicit use of B and M.

Binary Mergesort performs $O(N\log_2 N)$ comparisons, but analyzed in the cache-oblivious model it performs $O(\frac{N}{B}\log_2\frac{N}{M})$ block transfers which is a factor $\Theta(\log\frac{M}{B})$ from the lower bound (assuming a recursive implementation of binary Mergesort, in order to get M in the denominator in the $\log N/M$ part of the bound on the block transfers). Another comparison-based sorting algorithm is the classical Quicksort sorting algorithm from 1962 by Hoare [9], that performs expected $O(N\log_2 N)$ comparisons and expected $O(\frac{N}{B}\log_2\frac{N}{M})$ block transfers. Both these algorithms achieve their relatively good performance for the number of block transfers from the fact that they are based on repeated scanning of arrays—a property not shared with e. g. Heapsort [10] that has a very poor performance of $\Theta(N\log_2\frac{N}{M})$ block transfers. In the I/O model the optimal performance of $O(\frac{N}{B}\log_{M/B}\frac{N}{B})$ is achieved by generalizing binary Mergesort to $\Theta(\frac{M}{B})$-way Mergesort [1].

Frigo et al. in [8] presented two cache-oblivious sorting algorithms (which can also be used to permute an array of elements). The first algorithm [8, Section 4] is denoted *Funnelsort* and is a reminiscent of classical binary Mergesort, whereas the second algorithm [8, Section 5] is a distribution-based sorting algorithm. Both algorithms perform *optimal* $O(\frac{N}{B}\log_{M/B}\frac{N}{B})$ block transfers – provided a *tall cache assumption* $M = \Omega(B^2)$ is satisfied.

Funnelsort

The basic idea of Funnelsort is to rearrange the sorting process performed by binary Mergesort, such that the processed data is stored "locally." This is achieved by two basic ideas: (1) A top-level recursion that partitions the input into $N^{1/3}$ sequences of size $N^{2/3}$, Funnelsort these sequences recursively, and merge the resulting sorted subsequences using an $N^{1/3}$-*merger*. (2) A k-merger is recursively defined to perform binary merging of k input sequences in a clever schedule with an appropriate recursive layout of data in memory using buffers to hold suspended merging processes (see Fig. 1). Subsequently two simplifications were made, without sacrificing the asymptotic number of block transfers performed. In [3] it was proved that the binary merging could be performed lazily, simplifying the scheduling of merging. In [5] it was further observed that the recursive layout of k-mergers is not necessary. It is sufficient that a k-merger is stored in a consecutive array, i. e. the buffers can be laid out in arbitrary order which simplifies the construction algorithm for the k-merger.

Implicit Cache-Oblivious Sorting

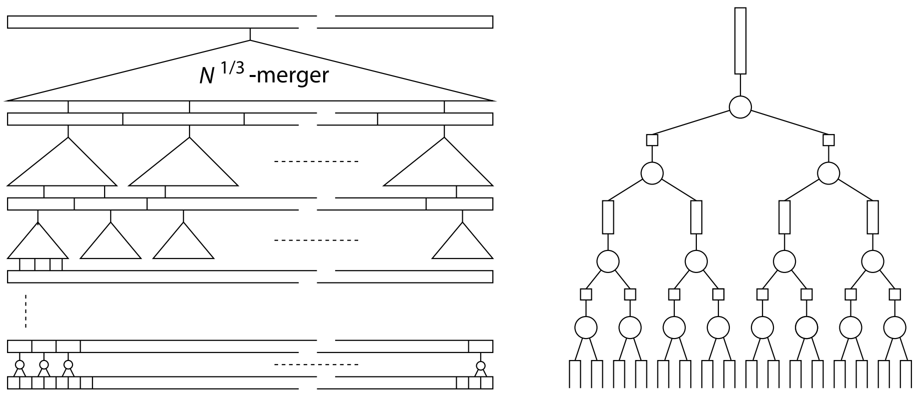
Franceschini in [7] showed how to perform optimal cache-oblivious sorting implicitly using only $O(1)$ space, i. e. all data is stored in the input array except for $O(1)$ additional words of information. In particular the output array is just a permutation of the input array.

The Role of the Tall Cache Assumption

The role of the tall cache assumption on cache-oblivious sorting was studied by Brodal and Fagerberg in [4]. If no tall cache assumption is made, they proved the following theorem:

Theorem 1 ([4], Corollary 3) *Let $B_1 = 1$ and $B_2 = M/2$. For any cache-oblivious comparison-based sorting algorithm, let t_1 and t_2 be upper bounds on the number of I/Os performed for block sizes B_1 and B_2. If for a real number $d \geq 0$ it is satisfied that $t_2 = d\cdot\frac{N}{B_2}\log_{M/B_2}\frac{N}{B_2}$ then $t_1 > 1/8\cdot N\log_2 N/M$.*

The theorem shows cache-oblivious comparison-based sorting without a tall cache assumption cannot match the performance of algorithms in the I/O model where M and



Cache-Oblivious Sorting, Figure 1
The overall recursion of Funnelsort (*left*) and a 16-merger (*right*)

B are known to the algorithm. It also has the natural interpretation that if a cache-oblivious algorithm is required to be I/O-optimal for the case $B = M/2$, then binary Mergesort is best possible –any other algorithm will be the same factor of $\Theta(\log M)$ worse than the optimal block transfer bound for the case $M \gg B$.

For the related problem of permuting an array the following theorem states that for all possible tall cache assumptions $B \leq M^\delta$, no cache-oblivious permuting algorithm exists with a block transfer bound (even only in the average case sense) matching the worst case bound in the I/O model.

Theorem 2 ([4], Theorem 2) *For all $\delta > 0$, there exists no cache-oblivious algorithm for permuting that for all $M \geq 2B$ and $1 \leq B \leq M^\delta$ achieves $O(\min\{N, \frac{N}{B} \log_{M/B} \frac{N}{B}\})$ I/Os averaged over all possible permutations of size N.*

Applications

Many problems can be reduced to cache-oblivious sorting. In particular Arge et al. [2] developed a cache-oblivious priority queue based on a reduction to sorting. They furthermore showed how a cache-oblivious priority queue can be applied to solve a sequence of graph problems, including list ranking, BFS, DFS, and minimum spanning trees.

Brodal and Fagerberg in [3] showed how to modify the cache-oblivious lazy Funnelsort algorithm to solve several problems within computational geometry, including orthogonal line segment intersection reporting, all nearest neighbors, 3D maxima problem, and batched orthogonal range queries. All these problems can be solved by a computation process very similarly to binary Mergesort with an additional problem dependent twist. This general framework to solve computational geometry problems is denoted *distribution sweeping*.

Open Problems

Since the seminal paper by Frigo et al. [8] introducing the cache-oblivious framework there has been a lot of work on developing algorithms with a good theoretical performance, but only a limited amount of work has been done on implementing these algorithms. An important issue for future work is to get further experimental results consolidating the cache-oblivious model as a relevant model for dealing efficiently with hierarchical memory.

Experimental Results

A detailed experimental study of the cache-oblivious sorting algorithm Funnelsort was performed in [5]. The main result of [5] is that a carefully implemented cache-oblivious sorting algorithm can be faster than a tuned implementation of Quicksort already for input sizes well within the limits of RAM. The implementation is also at least as fast as the recent cache-aware implementations included in the test. On disk the difference is even more pronounced regarding Quicksort and the cache-aware algorithms, whereas the algorithm is slower than a careful implementation of multiway Mergesort optimized for external memory such as in TPIE [6].

URL to Code

http://kristoffer.vinther.name/projects/funnelsort/

Cross References

▶ Cache-Oblivious Model
▶ External Sorting and Permuting
▶ I/O-model

Recommended Reading

1. Aggarwal, A., Vitter, J.S.: The input/output complexity of sorting and related problems. Commun. ACM **31**(9), 1116–1127 (1988)
2. Arge, L., Bender, M.A., Demaine, E.D., Holland-Minkley, B., Munro, J.I.: Cache-oblivious priority queue and graph algorithm applications. In: Proc. 34th Annual ACM Symposium on Theory of Computing, pp. 268–276. ACM Press, New York (2002)
3. Brodal, G.S., Fagerberg, R.: Cache oblivious distribution sweeping. In: Proc. 29th International Colloquium on Automata, Languages, and Programming. Lecture Notes in Computer Science, vol. 2380, pp. 426–438. Springer, Berlin (2002)
4. Brodal, G.S., Fagerberg, R.: On the limits of cache-obliviousness. In: Proc. 35th Annual ACM Symposium on Theory of Computing, pp. 307–315. ACM Press, New York (2003)
5. Brodal, G.S., Fagerberg, R., Vinther, K.: Engineering a cache-oblivious sorting algorithm. ACM J. Exp. Algoritmics (Special Issue of ALENEX 2004) **12**(2.2), 23 (2007)
6. Department of Computer Science, Duke University. TPIE: a transparent parallel I/O environment. http://www.cs.duke.edu/TPIE/. Accessed 2002
7. Franceschini, G.: Proximity mergesort: Optimal in-place sorting in the cache-oblivious model. In: Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), SIAM, p. 291. Philadelphia, 2004
8. Frigo, M., Leiserson, C.E., Prokop, H., Ramachandran, S.: Cache-oblivious algorithms. In: Proc. 40th Annual Symposium on Foundations of Computer Science, pp. 285–297. IEEE Computer Society Press, Los Alamitos (1999)
9. Hoare, C.A.R.: Quicksort. Comput. J. **5**(1), 10–15 (1962)
10. Williams, J.W.J.: Algorithm 232: Heapsort. Commun. ACM **7**(6), 347–348 (1964)

Caching

▶ Online Paging and Caching
▶ Paging

Causal Order, Logical Clocks, State Machine Replication

1978; Lamport

Xavier Défago
School of Information Science, Japan Advanced Institute of Science and Technology (JAIST),
Ishikawa, Japan

Keywords and Synonyms

State-machine replication: active replication

Problem Definition

This entry covers several problems, related with each other. The first problem is concerned with maintaining the causal relationship between events in a distributed system. The motivation is to allow distributed systems to reason about time with no explicit access to a physical clock. Lamport [5] defines a notion of logical clocks that can be used to generate timestamps that are consistent with causal relationships (in a conservative sense). He illustrates logical clocks (also called Lamport clocks) with a distributed mutual exclusion algorithm. The algorithm turns out to be an illustration of state-machine replication. Basically, the algorithm generates a total ordering of the events that is consistent across processes. With all processes starting in the same state, they evolve consistently with no need for further synchronization.

System Model

The system consists of a collection of processes. Each process consists of a sequence of events. Processes have no shared memory and communicate only by exchanging messages. The exact definition of an event depends on the system actually considered and the abstraction level at which it is considered. One distinguishes between three kinds of events: internal (affects only the process executing it), send, and receive events.

Causal Order

Causal order is concerned with the problem that the occurrence of some events may affect other events in the future, while other events may not influence each other. With processes that do not measure time, the notion of simultaneity must be redefined in such a way that simultaneous events are those that cannot possibly affect each other. For this reason, it is necessary to define what it means for an event to happen before another event.

The following "happened before" relation is defined as an irreflexive partial ordering on the set of all events in the system [5].

Definition 1 The relation "$\rightarrow$" on the set of events of a system is the smallest relation satisfying the following three conditions:

1. If a and b are events in the same process, and a comes before b, then $a \rightarrow b$.

2. If a is the sending of a message by one process and b is the receipt of the same message by another process, then $a \rightarrow b$.
3. If $a \rightarrow b$ and $b \rightarrow c$ then $a \rightarrow c$.

Definition 2 Two distinct events a and b are said to be *concurrent* if $a \not\rightarrow b$ and $b \not\rightarrow a$.

Logical Clocks

Lamport also defines clocks in a generic way, as follows.

Definition 3 A clock C_i for a process p_i is a function which assigns a number $C_i\langle a\rangle$ to any event a on that process. The entire system of clocks is represented by the function C which assigns to any event b the number $C\langle b\rangle$, where $C\langle b\rangle = C_j\langle b\rangle$ if b is an event in process p_j. The system of clocks must meet the following *clock condition*.
- For any events a and b, if $a \rightarrow b$ then $C\langle a\rangle < C\langle b\rangle$.

Assuming that there is some arbitrary total ordering $\prec$ of the processes (e. g., unique names ordered lexicographically), Lamport extends the "happened before" relation and defines a relation "$\Rightarrow$" as a total ordering on the set of all events in the system.

Definition 4 The total order relation $\Rightarrow$ is defined as follows. If a is an event in process p_i and b is an event in process p_j, then $a \Rightarrow b$ if and only if either one of the following conditions is satisfied.
1. $C_i\langle a\rangle < C_j\langle b\rangle$
2. $C_i\langle a\rangle = C_j\langle b\rangle$ and $p_i \prec p_j$.

In fact, Lamport [5] also discusses an adaptation of these conditions to physical clocks, and provides a simple clock synchronization algorithm. This is however not discussed further here.

State Machine Replication

The problem of state-machine replication was originally presented by Lamport [4,5]. In a later review of the problem, Schneider [8] defines the problem as follows (formulation adapted to the context of the entry).

Problem 1 (State-machine replication)
INPUT: *A set of concurrent requests.*
OUTPUT: *A sequence of the requests processed at each process, such that:*
1. *Replica coordination: all replicas receive and process the same sequence of requests.*
2. *Agreement: every non-faulty state-machine replica receives every request.*
3. *Order: every non-faulty state-machine replica processes the requests it receives in the same relative order.*

In his paper on logical time [5] and discussed in this entry, Lamport does not consider failures. He does however consider them in another paper on state-machine replication for fault-tolerance [4], which he published the same year.

Key Results

Lamport [5] proposed many key results related to the problems described above.

Logical Clocks

Lamport [5] defines an elegant system of logical clocks that meets the clock condition presented in Definition 3. The clock of a process p_i is represented by a register C_i, such that $C_i\langle a\rangle$ is the value held by C_i when a occurs. Each message m carries a timestamp T_m, which equals the time at which m was sent. The clock system can be described in terms of the following rules.
1. Each process p_i increments C_i between any two successive events.
2. If event a is the sending of a message m by process p_i, then the message m contains a timestamp $T_m = C_i\langle a\rangle$.
3. Upon receiving a message m, process p_j sets C_j to $\max(C_j, T_m + 1)$ (before actually executing the receive event).

State Machine Replication

As an illustration for the use of logical clocks, Lamport [5] describes a mutual exclusion algorithm. He also mentions that the approach is more general and discusses the concept of state-machine replication that he refines in a different paper [4].

The mutual exclusion algorithm is based on the idea that every process maintains a copy of a request queue, and the algorithm ensures that the copies remain consistent across the processes. This is done by generating a total ordering of the request messages, according to timestamps obtained from the logical clocks of the sending processes.

The algorithm described works under the following simplifying assumptions:
- Every message that is sent is eventually received.
- For any processes p_i and p_j, messages from p_i to p_j are received in the same order as they are sent.
- A process can send messages directly to every other processes.

The algorithm requires that each process maintains its own request queue, and ensures that the request queues of different processes always remain consistent. Initially, request queues contain a single message $(T_0, p_0, \mathit{request})$, where p_0 is the process that holds the resource and the timestamp T_0 is smaller than the initial value of every clock. Then, the algorithm works as follows.

1. When a process p_i requests the resource, it sends a request message $(T_m, p_i, \mathit{request})$ to all other processes and puts the message in its request queue.
2. When a process p_j receives a message $(T_m, p_i, \mathit{request})$, it puts that message in its request queue and sends an acknowledgment $(T_{m'}, p_j, \mathit{ack})$ to p_i.
3. When a process p_i releases the resource, it removes all instances of messages $(-, p_i, \mathit{request})$ from its queue, and sends a message $(T_{m'}, p_i, \mathit{release})$ to all other processes.
4. When a process p_j receives a release message from process p_i, it removes all instances of messages $(-, p_i, \mathit{request})$ from its queue, and sends a timestamped acknowledgment to p_i.
5. Messages in the queue are sorted according to the total order relation $\Rightarrow$ of Definition 4. A process p_i can use the resource when (a) a message $(T_m, p_i, \mathit{request})$ appears first in the queue, and (b) p_i has received from all other processes a message with a timestamp greater than T_m (or equal from any process p_j where $p_i \prec p_j$).

Applications

A brief overview of some applications of the concepts presented in this entry has been provided.

First of all, the notion of causality in distributed systems (or lack thereof) leads to a famous problem in which a user may potentially see an answer before she can see the relevant question. The time-independent characterization of causality of Lamport lead to the development of efficient solutions to enforce causal order in communication. In his later work, Lamport [3] gives a more general definition to the "happened before" relation, so that a system can be characterized at various abstraction levels.

About a decade after Lamport's work on logical clock, Fidge [2] and Mattern [6] have developed the notion of vector clocks, with the advantage of a complete characterization of causal order. Indeed, the clock condition enforced by Lamport's logical clocks is only a one-way implication (see Definition 3). In contrast, vector clocks extend Lamport clocks by ensuring that, for any events a and b, if $C\langle a\rangle < C\langle b\rangle$, then $a \rightarrow b$. This is for instance useful for choosing a set of checkpoints after recovery of a distributed system, for distributed debugging, or for deadlock detection. Other extensions of logical time have been proposed, that have been surveyed by Raynal and Singhal [7].

The state-machine replication also has many applications. In particular, it is often used for replicating a distributed service over several processors, so that the service can continue to operate even in spite of the failure of some of the processors. State-machine replication ensures that the different replicas remain consistent.

The mutual exclusion algorithm proposed by Lamport [5] and described in this entry is actually one of the first known solution to the *atomic broadcast* problem (see relevant entry). Briefly, in a system with several processes that broadcast messages concurrently, the problem requires that all processes deliver (and process) all message in the same order. Nowadays, there exist several approaches to solving the problem. Surveying many algorithms, Défago et al. [1] have classified Lamport's algorithm as *communication history* algorithms, because of the way the ordering is generated.

Cross References

► Atomic Broadcast
► Clock Synchronization
► Concurrent Programming, Mutual Exclusion
► Linearizability
► Quorums

Recommended Reading

1. Défago, X., Schiper, A., Urbán, P.: Total order broadcast and multicast algorithms: Taxonomy and survey. ACM Comput. Surv. **36**, 372–421 (2004)
2. Fidge, C. J.: Logical time in distributed computing systems. IEEE Comput. **24**, 28–33 (1991)
3. Lamport, L.: On interprocess communication. Part I: Basic formalism. Distrib. Comput. **1**, 77–85 (1986)
4. Lamport, L.: The implementation of reliable distributed multiprocess systems. Comput. Netw. **2**, 95–114 (1978)
5. Lamport, L.: Time, clocks, and the ordering of events in a distributed system. Commun. ACM **21**, 558–565 (1978)
6. Mattern, F.: Virtual time and global states of distributed systems. In: Cosnard, M., Quinton, P. (eds.) Parallel and Distributed Algorithms, pp.215–226. North-Holland, Amsterdam (1989)
7. Raynal, M., Singhal, M.: Capturing causality in distributed systems. IEEE Comput. **29**, 49–56 (1996)
8. Schneider, F.B.: Implementing fault-tolerant services using the state machine approach: a tutorial. ACM Comput. Surv. **22**, 299–319 (1990)

Certificate Complexity and Exact Learning

1995; Hellerstein, Pilliapakkamnatt, Raghavan, Wilkins

LISA HELLERSTEIN
Department of Computer and Information Science, Polytechnic University, Brooklyn, NY, USA

Problem Definition

This problem concerns the query complexity of proper learning in a widely studied learning model: exact learning with membership and equivalence queries. Hellerstein et al. [8] showed that the number of (polynomially sized) queries required to learn a concept class in this model is closely related to the size of certain certificates associated with that class. This relationship gives a combinatorial characterization of the concept classes that can be learned with polynomial query complexity. (Similar results were shown by Hegedüs[7], based on the work of Moshkov [11].)

The Exact Learning Model

Concepts are functions $f : X \rightarrow Y$ where X is an arbitrary domain, and $Y = \{0, 1\}$. In exact learning, there is a hidden concept f from a known class of concepts C, and the problem is to exactly identify the function f.

Algorithms in the exact learning model obtain information about f, the target concept, by querying two oracles, a membership oracle and an equivalence oracle. A membership oracle for f answers membership queries, which are of the form "What is $f(x)$?" where $x \in X$ (point evaluation queries). The membership oracle responds with the value $f(x)$. An equivalence oracle for f answers equivalence queries, which are of the form "Is $h \equiv f$?" where h is a representation of a concept defined on the domain X. Representation h is called a hypothesis. The equivalence oracle responds "yes" if $h(x) = f(x)$ for all $x \in X$. Otherwise, it returns a counterexample, a value $x \in X$ such that $f(x) \neq h(x)$.

The exact learning model is due to Angluin [2]. Angluin viewed the combination of membership and equivalence oracles as constituting a "minimally adequate teacher." Equivalence queries can be simulated both in Valiant's well-known PAC model, and in the on-line mistake-bound learning model.

Let R be a set of representations of concepts, and let C_R be the associated set of concepts. For example, if R were a set of DNF formulas, then C_R would be the set of Boolean functions (concepts) represented by those formulas. An exact learning algorithm is said to learn R if, given access to membership and equivalence oracles for any f in C_R, it ends by outputting a hypothesis h that is a representation of f.

Query Complexity of Exact Learning

There are two aspects to the complexity of exact learning, query complexity and computational complexity. The results of Hellerstein et al. concern query complexity.

The query complexity of an exact learning algorithm measures the number of queries it asks and the size of the hypotheses it uses in those queries (and as the final output). We assume that each representation class R has an associated size function that assigns a non-negative number to each $r \in R$. The size of a concept c with respect to R, denoted by $|c|_R$, is the size of the smallest representation of c in R; if $c \notin c_R$, $|c|_R = \infty$. Ideally, the query complexity of an exact learning algorithm will be polynomial in the size of the target and other relevant parameters of the problem.

Many exact learning results concern learning representations of Boolean functions. Algorithms for learning such classes R are said to have polynomial query complexity if the number of hypotheses used, and the size of those hypotheses, is bounded by some polynomial $p(m, n)$, where n is the number of variables on which the target f is defined, and $m = |f|_R$. We assume that algorithms for learning Boolean representation classes are given the value of n as input.

Since the number and size of queries used by an algorithm is a lower bound on the time taken by that algorithm, query complexity lower bounds imply computational complexity lower bounds.

Improper Learning and the Halving Algorithm

An algorithm for learning a representation class R is said to be proper if all hypotheses used in its equivalence queries are from R_C, and it outputs a representation from R_C. Otherwise, the algorithm is said to be improper.

When R_C is a finite concept class, defined on a finite domain X, a simple, generic algorithm called the halving algorithm can be used to exactly learn R using $\log |R_C|$ equivalence queries and no membership queries. The halving algorithm is based on the following idea. For any $V \subseteq R_C$, define the majority hypothesis MAJ_V to be the concept defined on X such that for all $x \in X$, $MAJ_V(x) = 1$ if $g(x) = 1$ for more than half the concepts g in V, and $MAJ_V(x) = 0$ otherwise. The halving algorithm begins by setting $V = R_C$. It then repeats the following:

1. Ask an equivalence query with the hypothesis MAJ_V.
2. If the answer is yes, then output MAJ_V.
3. Otherwise, the answer is a counterexample x. Remove from V all g such that $g(x) = MAJ_V(x)$.

Each counterexample eliminates the majority of the elements currently in V, so the size of V is reduced by a factor of at least 2 with each equivalence query. It follows that the algorithm cannot ask more than $\log_2 |R_C|$ queries.

The halving algorithm cannot necessarily be implemented as a proper algorithm, since the majority hypotheses may not be representable in R_C. Even when they are representable in R_C, the representations may be exponentially larger than the target concept.

Proper Learning and Certificates

In the exact model, the query complexity of proper learning is closely related to the size of certain certificates.

For any concept f defined on a domain X, a certificate that f has property P is a subset $S \subseteq X$ such that for all concepts g defined on X, if $g(x) = f(x)$ for all $x \in X$, then g has property P. The size of the certificate S is $|S|$, the number of elements in it.

We are interested in properties of the form "g is not a member of the concept class C". To take a simple example, let D be the class of constant-valued n-variable Boolean functions, i. e. D consists of the two functions $f_1(x_1, \dots, x_n) = 1$ and $f_2(x_1, \dots, x_n) = 0$. Then if g is an n-variable Boolean function that is not a member of D, a certificate that g is not in C could be just a pair $a \in \{0,1\}^n$ and $b \in \{0,1\}^n$ such that $g(a) = 1$ and $g(b) = 0$.

For C a class of concepts defined on X, define the exclusion dimension of C to be the maximum, over all concepts g not in C, of the size of the smallest certificate that g is not in C. Let $XD(C)$ denote the exclusion dimension of C. In the above example, $XD(C) = 2$.

Key Results

Theorem 1 *Let R be a finite class of representations. Then there exists a proper learning algorithm in the exact model that learns C using at most $XD(C) \log |C|$ queries. Further, any such algorithm for C must make at least $XD(C)$ queries.*

Independently, Hegedüs proved a theorem that is essentially identical to the above theorem. The algorithm in the theorem is a variant of the ordinary halving algorithm. As noted by Hegedüs, a similar result to Theorem 1 was proved earlier by Moshkov, and Moshkov's techniques can be used to improve the upper bound by a factor of $1/XD(C)$.

An extension of the above result characterizes the representation classes that have polynomial query complexity. The following theorem presents the extended result as it applies to representation classes of Boolean functions.

Theorem 2 *Let R be a class of representations of Boolean functions. Then there exists a proper exact learning algorithm for R with polynomial query complexity iff there exists a polynomial $p(m, n)$ such that for all $m, n > 0$, and all n-variable Boolean functions g, if the size of g is greater than m, then there exists a certificate of size at most $p(m, n)$ proving that $|g|_R > m$.*

A concept class having certificates of the type specified in this theorem is said to have polynomial certificates.

The algorithm in the above theorem does not run in polynomial time. Hellerstein et al. give a more complex algorithm that runs in polynomial time using a Σ_4^P oracle, provided R satisfies certain technical conditions. Köbler and Lindner subsequently gave an algorithm using a Σ_2^P oracle [10].

Theorem 2 and its generalization give a technique for proving bounds on proper learning in the exact model. Proving upper bounds on the size of the appropriate certificates yields upper bounds on query complexity. Proving lower bounds on the size of appropriate certificates yields lower bounds on query complexity and hence also on time complexity. Moreover, unlike many computational hardness results in learning, computational hardness results achieved in this way do not rely on any unproven complexity theoretic or cryptographic hardness assumptions.

One of the most widely studied problems in computational learning theory has been the question of whether DNF formulas can be learned in polynomial time in common learning models. The following result on learning DNF formulas was proved using Theorem 2, by bounding the size of the relevant certificates.

Theorem 3 *There is a proper algorithm that learns DNF formulas in the exact model with query complexity bounded above by a polynomial $p(m, r, n)$, where m is the size of the smallest DNF representing the target function f, n is the number of variables on which f is defined, and r is the size of the smallest CNF representing f.*

The size of a DNF is the number of its terms; the size of a CNF is the number of its clauses. The above theorem does not imply polynomial-time learnability of arbitrary DNF formulas, since the running time of the algorithm depends not just on the size of the smallest DNF representing the target, but also on the size of the smallest CNF.

Building on results of Alekhnovich et al., Feldman showed that if NP $\neq$ RP, DNF formulas cannot be learned in polynomial time in the PAC model augmented with membership queries. The same negative result then follows immediately for the exact model [1,6]. Hellerstein and Raghavan used certificate size lower bounds and Theorem 1 to prove that DNF formulas cannot be learned by a proper exact algorithm with polynomial query complexity, if the algorithm is restricted to using DNF hypotheses that are only slightly larger than the target [9].

The main results of Hellerstein et al. apply to learning with membership and equivalence queries. Hellerstein et al. also considered the model of exact learning with membership queries alone, and showed that in this model, a projection-closed Boolean function class is polynomial query learnable iff it has polynomial teaching dimension. Teaching dimension was previously defined by Goldman and Kearns. Hegedüs defined the extended teaching dimension, and showed that all classes are polynomially query learnable with membership queries alone iff they have polynomial extended teaching dimension.

Balcázar et al. introduced the strong consistency dimension to characterize polynomial query learnability with equivalence queries alone [5]. The abstract identification dimension of Balcázar, Castro, and Guijarro is a very general measure that can be used to characterize polynomial query learnability for any set of example-based queries [4].

Applications

None

Open Problems

It remains open whether DNF formulas can be learned in polynomial time in the exact model, using hypotheses that are not DNF formulas.

Feldman's results show the computational hardness of proper learning of DNF in the exact learning model based on complexity theoretic assumptions. However, it is unclear whether the hardness of proper learning of DNF is a result of computational barriers, or whether query complexity is also a barrier to efficient learning. It is still open whether the class of DNF formulas has polynomial certificates; showing they do not have polynomial certificates would give a hardness result for proper learning of DNF based only on query complexity, with no complexity theoretic assumptions (and without the hypothesis-size restrictions used by Hellerstein and Raghavan). It is also open whether the class of Boolean decision trees has polynomial certificates.

Certificate techniques are used to prove lower bounds on learning when we restrict the type of hypotheses used by the learning algorithm. These types of results are called representation dependent, since they depend on the restriction of the representations used as hypotheses. Although there are some techniques for proving representation-independent hardness results, there is a need for more powerful techniques.

Recommended Reading

1. Alekhnovich, M., Braverman, M., Feldman, V., Klivans, A.R., Pitassi, T.: Learnability and automatizability. In: FOCS '04 Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS'04), pp. 621–630. IEEE Computer Society, Washington (2004)
2. Angluin, D.: Queries and concept learning. Mach. Learn. **2**(4), 319–342 (1987)
3. Angluin, D.: Queries Revisited. Theor. Comput. Sci. **313**(2), 175–194 (2004)
4. Balcázar, J.L., Castro, J., Guijarro, D.: A new abstract combinatorial dimension for exact learning via queries. J. Comput. Syst. Sci. **64**(1), 2–21 (2002)
5. Balcázar, J.L., Castro, J., Guijarro, D., Simon, H.-U.: The consistency dimension and distribution-dependent learning from queries. Theor. Comput. Sci. **288**(2), 197–215 (2002)
6. Feldman, V.: Hardness of approximate two-level logic minimization and pac learning with membership queries. In: STOC '06 Proceedings of the 38th Annual ACM Symposium on Theory of Computing, pp. 363–372. ACM Press, New York (2006)
7. Hegedüs, T.: Generalized teaching dimensions and the query complexity of learning. In: COLT '95 Proceedings of the 8th Annual Conference on Computational Learning Theory, pp. 108–117 (1995)
8. Hellerstein, L., Pillaipakkamnatt, K., Raghavan, V., Wilkins, D.: How many queries are needed to learn? J. ACM. **43**(5), 840–862 (1996)
9. Hellerstein, L., Raghavan, V.: Exact learning of dnf formulas using dnf hypotheses. J Comput. Syst. Sci. **70**(4), 435–470 (2005)
10. Köbler, J., Lindner, W.: Oracles in $s^p{}_2$ are sufficient for exact learning. Int. J. Found. Comput. Sci. **11**(4), 615–632 (2000)
11. Moshkov, M.Y.: Conditional tests. Probl. Kibern. (in Russian) **40**, 131–170 (1983)

Channel Assignment and Routing in Multi-Radio Wireless Mesh Networks

2005; Alicherry, Bhatia, Li

MANSOOR ALICHERRY, RANDEEP BHATIA,
LI (ERRAN) LI
Bell Labs, Alcatel-Lucent, Murray Hill, NJ, USA

Keywords and Synonyms

Graph coloring, Ad-hoc networks

Problem Definition

One of the major problems facing wireless networks is the capacity reduction due to interference among multiple simultaneous transmissions. In wireless mesh networks providing mesh routers with multiple-radios can greatly alleviate this problem. With multiple-radios, nodes can transmit and receive simultaneously or can transmit on multiple channels simultaneously. However, due to the limited number of channels available the interference cannot be completely eliminated and in addition careful channel assignment must be carried out to mitigate the effects of interference. Channel assignment and routing are inter-dependent. This is because channel assignments have an impact on link bandwidths and the extent to which link transmissions interfere. This clearly impacts the routing used to satisfy traffic demands. In the same way traffic routing determines the traffic flows for each link which certainly affects channel assignments. Channel assignments need to be done in a way such that the communication requirements for the links can be met. Thus, the problem of throughput maximization of wireless mesh networks must be solved through channel assignment, routing, and scheduling.

Formally, given a wireless mesh backbone network modeled as a graph (V, E): The node $t \in V$ represents the wired network. An edge $e = (u, v)$ exists in E iff u and v are within *communication range* R_T. The set $V_G \subseteq V$ represents the set of gateway nodes. The system has a total of K channels. Each node $u \in V$ has $I(u)$ network interface cards, and has an aggregated demand $l(u)$ from its associated users. For each edge e the set $I(e) \subset E$ denotes the set of edges that it interferes with. A pair of nodes that use the same channel and are within *interference range* $R_I x$ may interfere with each other's communication, even if they cannot directly communicate. Node pairs using different channels can transmit packets simultaneously without interference. The problem is to maximize λ where at least $\lambda l(u)$ amount of throughput can be routed from each node u to the Internet (represented by a node t). The $\lambda l(u)$ throughput for each node u is achieved by computing $g(1)$ a network flow that associates with each edge $e = (u, v)$ values $f(e(i)), 1 \leq i \leq K$ where $f(e(i))$ is the rate at which traffic is transmitted by node u for node v on channel i; (2) a feasible channel assignment $F(u)$ ($F(u)$ is an ordered set where the ith interface of u operates on the ith channel in $F(u)$) such that, whenever $f(e(i)) > 0$, $i \in F(u) \cap F(v)$; (3) a *feasible* schedule S that decides the set of edge channel pair (e, i) (edge e using channel, i. e. $f(e(i)) > 0$ scheduled at time slot τ, for $\tau = 1, 2, \ldots, T$ where T is the period of the schedule. A schedule is feasible if the edges of no two edge pairs $(e_1, i), (e_2, i)$ scheduled in the same time slot for a common channel i interfere with each other ($e_1 \notin I(e_2)$ and $e_2 \notin I(e_1)$). Thus, a feasible schedule is also referred to as an interference free edge schedule. An indicator variable $X_{e,i,\tau}$, $e \in E$, $i \in F(e)$, $\tau \geq 1$ is used. It is assigned 1 if and only if link e is active in slot τ on channel i. Note that $1/T \sum_{1 \leq \tau \leq T} X_{e,i,\tau} c(e) = f(e(i))$. This is because communication at rate $c(e)$ happens in every slot that link e is active on channel i and since $f(e(i))$ is the average rate attained on link e for channel i. This implies $1/T \sum_{1 \leq \tau \leq T} X_{e,i,\tau} = \frac{f(e(i))}{c(e)}$.

Joint Routing, Channel Assignment, and Link Scheduling Algorithm

Even the interference free edge scheduling sub-problem given the edge flows is NP-hard [5]. An approximation algorithm called RCL for the joint routing, channel assignment, and link scheduling problem has been developed. The algorithm performs the following steps in the given order:

1. **Solve LP:** First optimally solve a LP relaxation of the problem. This results in a flow on the flow graph along with a not necessarily feasible channel assignment for the node radios. Specifically, a node may be assigned more channels than the number of its radios. However, this channel assignment is "optimal" in terms of ensuring that the interference for each channel is minimum. This step also yields a lower bound on the λ value which is used in establishing the worst case performance guarantee of the overall algorithm.
2. **Channel Assignment:** This step presents a channel assignment algorithm which is used to adjust the flow on the flow graph (routing changes) to ensure a feasible channel assignment. This flow adjustment also strives to keep the increase in interference for each channel to a minimum.
3. **Interference Free Link Scheduling:** This step obtains an interference free link schedule for the edge flows corresponding to the flow on the flow graph.

Each of these steps is described in the following subsections.

A Linear Programming-Based Routing Algorithm

A linear program LP (1) to find a flow that maximizes λ is given below:

$$\max \lambda \tag{1}$$

Subject to

$$\lambda l(v) + \sum_{e=(u,v)\in E} \sum_{i=1}^{K} f(e(i)) = \sum_{e=(v,u)\in E} \sum_{i=1}^{K} f(e(i)) , \quad \forall v \in V - V_G \tag{2}$$

$$f(e(i)) \leq c(e) , \quad \forall e \in E \tag{3}$$

$$\sum_{1\le i\le K}\left(\sum_{e=(u,v)\in E}\frac{f(e(i))}{c(e)}+\sum_{e=(v,u)\in E}\frac{f(e(i))}{c(e)}\right)\le I(v)\,,\qquad v\in V \quad (4)$$

$$\frac{f(e(i))}{c(e)}+\sum_{e'\in I(e)}\frac{f(e'(i))}{c(e')}\le c(q)\,,\qquad \forall e\in E\,,\ 1\le i\le K\,. \quad (5)$$

The first two constraints are *flow constraints*. The first one is the flow conservation constraint; the second one ensures no link capacity is violated. The third constraint is the *node radio constraints*. Recall that a IWMN node $v \in V$ has $I(v)$ radios and hence can be assigned at most $I(v)$ channels from $1 \le i \le K$. One way to model this constraint is to observe that due to interference constraints v can be involved in at most $I(v)$ simultaneous communications (with different one hop neighbors). In other words this constraint follows from $\sum_{1\le i\le K}\sum_{e=(u,v)\in E}X_{e,i,\tau} + \sum_{1\le i\le K}\sum_{e=(v,u)\in E}X_{e,i,\tau} \le I(v)$. The fourth constraint is the *link congestion constraints* which are discussed in detail in Sect. "Link Flow Scheduling". Note that all the constraints listed above are necessary conditions for any feasible solution. However, these constraints are not necessarily sufficient. Hence if a solution is found that satisfies these constraints it may not be a feasible solution. The approach is to start with a "good" but not necessarily feasible solution that satisfies all of these constraints and use it to construct a feasible solution without impacting the quality of the solution.

A solution to this LP can be viewed as a flow on a *flow graph* $H = (V, E^H)$ where $E^H = \{e(i)|\forall e \in E, 1 \le i \le K\}$. Although the optimal solution to this LP yields the best possible λ (say λ^*) from a practical point of view more improvements may be possible:

- The flow may have directed cycles. This may be the case since the LP does not try to minimize the amount of interference directly. By removing the flow on the directed cycle (equal amount off each edge) flow conservation is maintained and in addition since there are fewer transmissions the amount of interference is reduced.
- The flow may be using a long path when shorter paths are available. Note that longer paths imply more link transmissions. In this case it is often the case that by moving the flow to shorter paths, system interference may be reduced.

The above arguments suggest that it would be practical to find among all solutions that attain the optimal λ value of λ^* the one for which the total value of the following quantity is minimized:

$$\sum_{1\le i\le K}\sum_{e=(v,u)\in E}\frac{f(e(i))}{c(e)}\,.$$

The LP is then re-solved with this objective function and with λ fixed at λ^*.

Channel Assignment

The solution to the LP (1) is a set of flow values $f(e(i))$ for edge e and channel i that maximize the value λ. Let λ^* denote the optimal value of λ. The flow $f(e(i))$ implies a channel assignment where the two end nodes of edge e are both assigned channel i if and only if $f(e(i)) > 0$. Note that for the flow $f(e(i))$ the implied channel assignment may not be feasible (it may require more than $I(v)$ channels at node v). The channel assignment algorithm transforms the given flow to fix this infeasibility. Below only a sketch of the algorithm is given. More details can be found in [1].

First observe that in an idle scenario, where all nodes v have the same number of interfaces I (i. e. $I = I(v)$) and where the number of available channels K is also I, the channel assignment implied by the LP (1) is feasible. This is because even the trivial channel assignment where all nodes are assigned all the channels 1 to I is feasible. The main idea behind the algorithm is to first transform the LP (1) solution to a new flow in which every edge e has flow $f(e(i)) > 0$ only for the channels $1 \le i \le I$. The basic operation that the algorithm uses for this is to equally distribute, for every edge e, the flow $f(e(i))$, for $I < i \le K$ to the edges $e(j)$, for $1 \le i \le I$. This ensures that all $f(e(i)) = 0$, for $I < i \le K$ after the operation. This operation is called Phase I of the Algorithm. Note that the Phase I operation does not violate the flow conservation constraints or the node radio constraints (5) in the LP (1). It can be shown that in the resulting solution the flow $f(e(i))$ may exceed the capacity of edge e by at most a factor $\phi = K/I$. This is called the "inflation factor" of Phase I. Likewise in the new flow, the *link congestion constraints* (5) may also be violated for edge e and channel i by no more than the inflation factor φ. In other words in the resulting flow

$$\frac{f(e(i))}{c(e)}+\sum_{e'\in I(e)}\frac{f(e'(i))}{c(e')}\le \phi c(q)\,.$$

This implies that if the new flow is scaled by a fraction $1/\phi$ than it is feasible for the LP (1). Note that the implied channel assignment (assign channels 1 to I to every node) is also feasible. Thus, the above algorithm finds a feasible channel assignment with a λ value of at least λ^*/ϕ.

One shortcoming of the channel assignment algorithm (Phase I) described so far is that it only uses I of the K available channels. By using more channels the interference may be further reduced thus allowing for more flow to be pushed in the system. The channel assignment algorithm uses an additional heuristic for this improvement. This is called Phase II of the algorithm.

Now define an operation called "channel switch operation." Let A be a maximal connected component (the vertices in A are not connected to vertices outside A) in the graph formed by the edges e for a given channel i for which $f(e(i)) > 0$. The main observation to use is that for a given channel j, the operation of completely moving flow $f(e(i))$ to flow $f(e(j))$ for every edge e in A, does not impact the feasibility of the implied channel assignment. This is because there is no increase in the number of channels assigned per node after the flow transformation: the end nodes of edges e in A which were earlier assigned channel i are now assigned channel j instead. Thus, the transformation is equivalent to switching the channel assignment of nodes in A so that channel i is discarded and channel j is gained if not already assigned.

The Phase II heuristic attempts to re-transform the unscaled Phase I flows $f(e(i))$ so that there are multiple connected components in the graphs $G(e, i)$ formed by the edges e for each channel $1 \leq i \leq I$. This re-transformation is done so that the LP constraints are kept satisfied with an inflation factor of at most φ, as is the case for the unscaled flow after Phase I of the algorithm.

Next in Phase III of the algorithm the connected components within each graph $G(e, i)$ are grouped such that there are as close to K (but no more than) groups overall and such that the maximum interference within each group is minimized. Next the nodes within the lth group are assigned channel l, by using the channel switch operation to do the corresponding flow transformation. It can be shown that the channel assignment implied by the flow in Phase III is feasible. In addition the underlying flows $f(e(i))$ satisfy the LP (1) constraints with an inflation factor of at most $\phi = K/I$.

Next the algorithm scales the flow by the largest possible fraction (at least $1/\phi$) such that the resulting flow is a feasible solution to the LP (1) and also implies a feasible channel assignment solution to the channel assignment. Thus, the overall algorithm finds a feasible channel assignment (by not necessarily restricting to channels 1 to I only) with a λ value of at least λ^*/ϕ.

Link Flow Scheduling

The results in this section are obtained by extending those of [4] for the single channel case and for the Protocol Model of interference [2]. Recall that the time slotted schedule S is assumed to be periodic (with period T) where the indicator variable $X_{e,i,\tau}, e \in E, i \in F(e), \tau \geq 1$ is 1 if and only if link e is active in slot τ on channel i and i is a channel in common among the set of channels assigned to the end-nodes of edge e.

Directly applying the result (Claim 2) in [4] it follows that a necessary condition for interference free link scheduling is that for every $e \in E, i \in F(e), \tau \geq 1$: $X_{e,i,\tau} + \sum_{e' \in I(e)} X_{e',i,\tau} \leq c(q)$. Here $c(q)$ is a constant that only depends on the interference model. In the interference model this constant is a function of the fixed value q, the ratio of the interference range R_I to the transmission range R_T, and an intuition for its derivation for a particular value $q = 2$ is given below.

Lemma 1 $c(q) = 8$ *for* $q = 2$.

Proof Recall that an edge $e' \in I(e)$ if there exist two nodes $x, y \in V$ which are at most $2R_T$ apart and such that edge e is incident on node x and edge e' is incident on node y. Let $e = (u, v)$. Note that u and v are at most R_T apart. Consider the region C formed by the union of two circles C_u and C_v of radius $2R_T$ each, centered at node u and node v, respectively. Then $e' = (u', v') \in I(e)$ if an only if at least one of the two nodes u', v' is in C; Denote such a node by $C(e')$. Given two edges $e_1, e_2 \in I(e)$ that do not interfere with each other it must be the case that the nodes $C(e_1)$ and $C(e_2)$ are at least $2R_T$ apart. Thus, an upper bound on how many edges in $I(e)$ do not pair-wise interfere with each other can be obtained by computing how may nodes can be put in C that are pair-wise at least $2R_T$ apart. It can be shown [1] that this number is at most 8. Thus, in schedule S in a given slot only one of the two possibilities exist: either edge e is scheduled or an "independent" set of edges in $I(e)$ of size at most 8 is scheduled implying the claimed bound.□

A necessary condition: (*Link Congestion Constraint*) Recall that $\frac{1}{T}\sum_{1 \leq \tau \leq T} X_{e,i,\tau} = \frac{f(e(i))}{c(e)}$. Thus: Any valid "interference free" edge flows must satisfy for every link e and every channel i the Link Congestion Constraint:

$$\frac{f(e(i))}{c(e)} + \sum_{e' \in I(e)} \frac{f(e'(i))}{c(e')} \leq c(q). \qquad (6)$$

A matching sufficient condition can also established [1].

A sufficient condition: (*Link Congestion Constraint*) If the edge flows satisfy for every link e and every channel i

the following *Link Schedulability Constraint* than an interference free edge communication schedule can be found using an algorithm given in [1].

$$\frac{f(e(i))}{c(e)} + \sum_{e' \in I(e)} \frac{f(e'(i))}{c(e')} \leq 1 . \qquad (7)$$

The above implies that if a flow $f(e(i))$ satisfies the *Link Congestion Constraint* then by scaling the flow by a fraction $1/c(q)$ it can be scheduled free of interference.

Key Results

Theorem *The RCL algorithm is a $Kc(q)/I$ approximation algorithm for the Joint Routing and Channel Assignment with Interference Free Edge Scheduling problem.*

Proof Note that the flow $f(e(i))$ returned by the channel assignment algorithm in Sect. "Channel Assignment" satisfies the *Link Congestion Constraint*. Thus, from the result of Sect. "Link Flow Scheduling" it follows that by scaling the flow by an additional factor of $1/c(q)$ the flow can be realized by an interference free link schedule. This implies a feasible solution to the joint routing, channel assignment and scheduling problem with a λ value of at least $\lambda^*/\phi c(q)$. Thus, the RCL algorithm is a $\phi c(q) = Kc(q)/I$ approximation algorithm. □

Applications

Infrastructure mesh networks are increasingly been deployed for commercial use and law enforcement. These deployment settings place stringent requirements on the performance of the underlying IWMNs. Bandwidth guarantee is one of the most important requirements of applications in these settings. For these IWMNs, topology change is infrequent and the variability of aggregate traffic demand from each mesh router (client traffic aggregation point) is small. These characteristics admit periodic optimization of the network which may be done by a system management software based on traffic demand estimation. This work can be directly applied to IWMNs. It can also be used as a benchmark to compare against heuristic algorithms in multi-hop wireless networks.

Open Problems

For future work, it will be interesting to investigate the problem when routing solutions can be enforced by changing link weights of a distributed routing protocol such as OSPF. Also, can the worst case bounds of the algorithm be improved (e. g. a constant factor independent of K and I)?

Cross References

► Graph Coloring
► Stochastic Scheduling

Recommended Reading

1. Alicherry, M., Bhatia, R., Li, L.E.: Joint channel assignment and routing for throughput optimization in multi-radio wireless mesh networks. In: Proc. ACM MOBICOM 2005, pp. 58–72
2. Gupta, P., Kumar, P.R.: The Capacity of Wireless Networks. IEEE Trans. Inf. Theory, IT-**46**(2), 388–404 (2000)
3. Jain, K., Padhye, J., Padmanabhan, V.N., Qiu, L.: Impact of interference on multi-hop wireless network performance. In: Proc. ACM MOBICOM 2003, pp. 66–80
4. Kumar, V.S.A., Marathe, M.V., Parthasarathy, S., Srinivasan, A.: Algorithmic aspects of capacity in wireless networks. In: Proc. ACM SIGMETRICS 2005, pp. 133–144
5. Kumar, V.S.A., Marathe, M.V., Parthasarathy, S., Srinivasan, A.: End-to-end packet-scheduling in wireless ad-hoc networks. In: Proc. ACM-SIAM symposium on Discrete algorithms 2004, pp. 1021–1030
6. Kyasanur, P., Vaidya, N.: Capacity of multi-channel wireless networks: Impact of number of channels and interfaces. In: Proc. ACM MOBICOM, pp. 43–57. 2005

Circuit Partitioning: A Network-Flow-Based Balanced Min-Cut Approach

1994; Yang, Wong

HONGHUA HANNAH YANG[1], MARTIN D. F. WONG[2]
[1] Strategic CAD Labs, Intel Corporation, Hillsboro, USA
[2] Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Urbana, IL, USA

Keywords and Synonyms

Hypergraph partitioning; Netlist partitioning

Problem Definition

Circuit partitioning is a fundamental problem in many areas of VLSI layout and design. *Min-cut balanced bipartition* is the problem of partitioning a circuit into two disjoint components with equal weights such that the number of nets connecting the two components is minimized. The min-cut balanced bipartition problem was shown to be NP-complete [5]. The problem has been solved by heuristic algorithms, e. g., Kernighan and Lin type (K&L) iterative improvement methods [4,11], simulated annealing

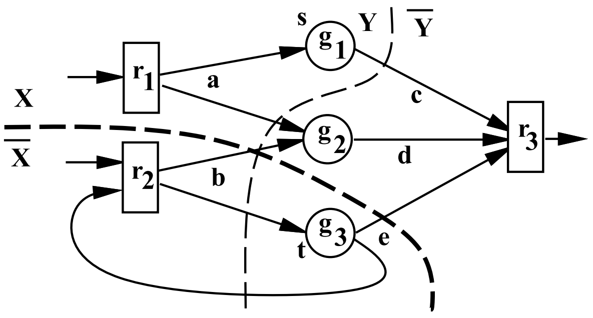
```
Algorithm: Flow-Balanced-Bipartition (FBB)
1. Pick a pair of nodes $s$ and $t$ in $N$;
2. Find a min-net-cut $C$ in $N$;
   Let $X$ be the subcircuit reachable from $s$ through
   augmenting paths in the flow network, and $\bar{X}$
   the rest;
3. if $(1-\epsilon)rW \le w(X) \le (1+\epsilon)rW$
     return $C$ as the answer;
4. if $w(X) < (1-\epsilon)rW$
4.1. Collapse all nodes in $X$ to $s$;
4.2. Pick a node $v \in \bar{X}$ adjacent to $C$ and collapse it to $s$;
4.3. Goto 1;
5. if $w(X) > (1+\epsilon)rW$
5.1. Collapse all nodes in $\bar{X}$ to $t$;
5.2. Pick a node $v \in X$ adjacent to $C$ and collapse it to $t$;
5.3. Goto 1;
```

Circuit Partitioning: A Network-Flow-Based Balanced Min-Cut Approach, Figure 1
FBB algorithm

```
Procedure: Incremental Flow Computation
1. while ∃ an additional augmenting path from $s$ to $t$
        increase flow value along the augmenting
        path;
2. Mark all nodes $u$ s.t. ∃ an augmenting path from $s$
   to $u$;
3. Let $C'$ be the set of bridging edges whose starting
   nodes are marked and ending nodes are not
   marked;
4. Return the nets corresponding to the bridging edges
   in $C'$ as the min-net-cut $C$, and the marked
   nodes as $X$.
```

Circuit Partitioning: A Network-Flow-Based Balanced Min-Cut Approach, Figure 2
Incremental max-flow computation

approaches [10], and analytical methods for the ratio-cut objective [2,7,13,15]. Although it is a natural method for finding a min-cut, the network max-flow min-cut technique [6,8] has been overlooked as a viable approach for circuit partitioning. In [16], a method was proposed for exactly modeling a circuit netlist (or, equivalently, a hypergraph) by a flow network, and an algorithm for balanced bipartition based on repeated applications of the max-flow min-cut technique was proposed as well. Our algorithm has the same asymptotic time complexity as one max-flow computation.



Circuit Partitioning: A Network-Flow-Based Balanced Min-Cut Approach, Figure 3
A circuit netlist with two net-cuts

A *circuit netlist* is defined as a digraph $N = (V, E)$, where V is a set of nodes representing logic gates and registers and E is a set of edges representing wires between gates and registers. Each node $v \in V$ has a weight $w(v) \in R^+$. The total weight of a subset $U \subseteq V$ is denoted by $w(U) = \Sigma_{v \in U} w(v)$. $W = w(V)$ denotes the total weight of the circuit. A *net* $n = (v; v_1, \ldots, v_l)$ is a set of outgoing edges from node v in N. Given two nodes s and t in N, an $s - t$ *cut* (or *cut* for short) $(X, \bar{X})$ of N is a bipartition of the nodes in V such that $s \in X$ and $t \in \bar{X}$. The *net-cut* $net(X, \bar{X})$ of the cut is the set of nets in N that are incident to nodes in both X and $\bar{X}$. A cut $(X, \bar{X})$ is a *min-net-cut* if $|net(X, \bar{X})|$ is minimum among all $s - t$ cuts of N. In Fig. 3, net $a = (r_1; g_1, g_2)$, net cuts $net(X, \bar{X}) = \{b, e\}$ and $net(Y, \bar{Y}) = \{c, a, b, e\}$, and $(X, \bar{X})$ is a min-net-cut.

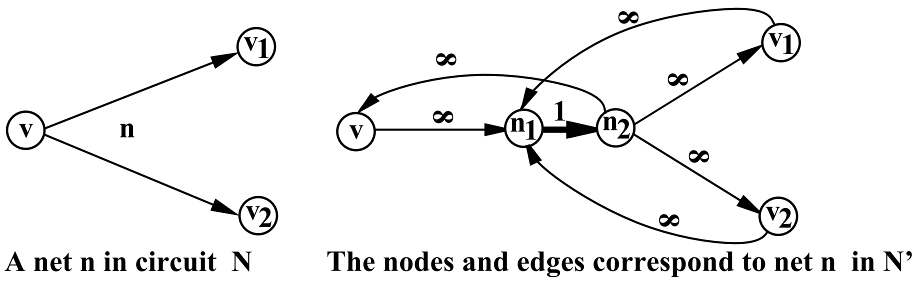
Formally, given an aspect ratio r and a deviation factor ϵ, *min-cut r-balanced bipartition* is the problem of finding a bipartition $(X, \bar{X})$ of the netlist N such that (1) $(1-\epsilon)rW \le W(X) \le (1+\epsilon)rW$ and (2) the size of the cut $net(X, \bar{X})$ is minimum among all bipartitions satisfying (1). When $r = 1/2$, this becomes a min-cut balanced-bipartition problem.

Key Results

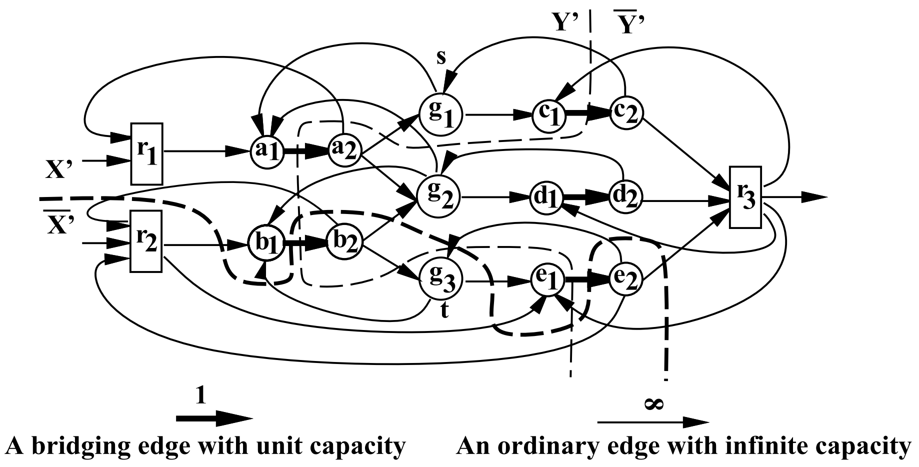
Optimal-Network-Flow-Based Min-Net-Cut Bipartition

The problem of finding a min-net-cut in $N = (V, E)$ is reduced to the problem of finding a cut of minimum capacity. Then the latter problem is solved using the max-flow min-cut technique. A flow network $N' = (V', E')$ is constructed from $N = (V, E)$ as follows (see Figs. 4 and 5):

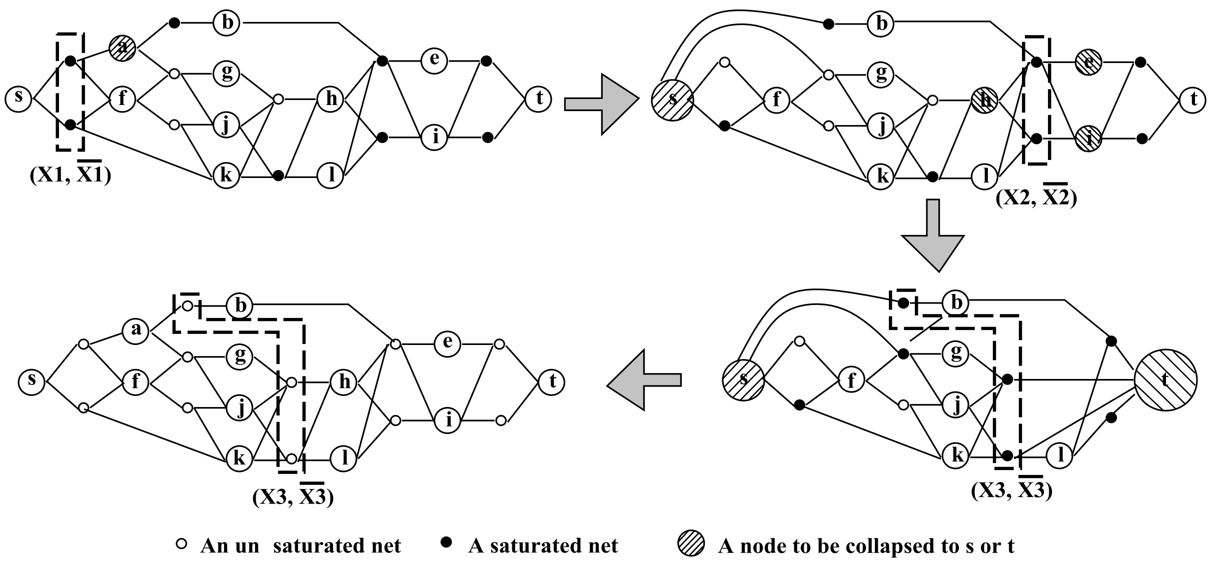
1. V' contains all nodes in V.
2. For each net $n = (v; v_1, \ldots, v_l)$ in N, add two nodes n_1 and n_2 in V' and a *bridging edge* $bridge(n) = (n_1, n_2)$ in E'.



Circuit Partitioning: A Network-Flow-Based Balanced Min-Cut Approach, Figure 4
Modeling a net in *N* in the flow network *N′*


Y'
Ȳ'
s
t
X'
X̄'
r1
r2
r3
a1
a2
b1
b2
g1
g2
g3
c1
c2
d1
d2
e1
e2
1
A bridging edge with unit capacity
∞
An ordinary edge with infinite capacity

Circuit Partitioning: A Network-Flow-Based Balanced Min-Cut Approach, Figure 5
The flow network for Fig. 3



Circuit Partitioning: A Network-Flow-Based Balanced Min-Cut Approach, Figure 6
FBB on the example in Fig. 5 for $r = 1/2$, $\epsilon = 0.15$ and unit weight for each node. The algorithm terminates after finding cut $(X_2, \bar{X}_2)$. A small solid node indicates that the bridging edge corresponding to the net is saturated with flow

Circuit Partitioning: A Network-Flow-Based Balanced Min-Cut Approach, Table 1
Comparison of SN, PFM3, and FBB ($r = 1/2, \epsilon = 0.1$)

Circuit				Avg. net-cut size			FBB bipart. ratio	Improve. %	
Name	Gates and latches	Nets	Avg. deg	SN	PFM3	FBB		Over SN	Over PFM3
C1355	514	523	3.0	38.9	29.1	26.0	1:1.08	33.2	10.7
C2670	1161	1254	2.6	51.9	46.0	37.1	1:1.15	28.5	19.3
C3540	1667	1695	2.7	90.3	71.0	79.8	1:1.11	11.6	−12.4
C7552	3466	3565	2.7	44.3	81.8	42.9	1:1.08	3.2	47.6
S838	478	511	2.6	27.1	21.0	14.7	1:1.04	45.8	30.0
Ave							1:1.10	24.5	19.0

Circuit Partitioning: A Network-Flow-Based Balanced Min-Cut Approach, Table 2
Comparison of EIG1, PB, and FBB ($r = 1/2, \epsilon = 0.1$). All allow $\leq$ 10% deviation

Circuit				Best net-cut size			Improve. % over		FBB elaps. sec.
Name	Gates and latches	Nets	Avg. deg	EIG1	PB	FBB	EIG1	PB	
S1423	731	743	2.7	23	16	13	43.5	18.8	1.7
S9234	5808	5805	2.4	227	74	70	69.2	5.4	55.7
S13207	8696	8606	2.4	241	91	74	69.3	18.9	100.0
S15850	10310	10310	2.4	215	91	67	68.8	26.4	96.5
S35932	18081	17796	2.7	105	62	49	53.3	21.0	2808
S38584	20859	20593	2.7	76	55	47	38.2	14.5	1130
S38417	24033	23955	2.4	121	49	58	52.1	−18.4	2736
Average							58.5	11.3	

3. For each node $u \in \{v, v_1, \ldots, v_l\}$ incident on net n, add two edges (u, n_1) and (n_2, u) in E'.
4. Let s be the source of N' and t the sink of N'.
5. Assign unit capacity to all bridging edges and infinite capacity to all other edges in E'.
6. For a node $v \in V'$ corresponding to a node in V, $w(v)$ is the weight of v in N. For a node $u \in V'$ split from a net, $w(u) = 0$.

Note that all nodes incident on net n are connected to n_1 and are connected from n_2 in N'. Hence the flow network construction is symmetric with respect to all nodes incident on a net. This construction also works when the netlist is represented as a hypergraph.

It is clear that N' is a strongly connected digraph. This property is the key to reducing the bidirectional min-net-cut problem to a minimum-capacity cut problem that counts the capacity of the forward edges only.

Theorem 1 *N has a cut of net-cut size at most C if and only if N' has a cut of capacity at most C.*

Corollary 1 *Let $(X', \bar{X}')$ be a cut of minimum capacity C in N'. Let $N_{cut} = \{n \mid bridge(n) \in (X', \bar{X}')\}$. Then $N_{cut} = (X, \bar{X})$ is a min-net-cut in N and $|N_{cut}| = C$.*

Corollary 2 *A min-net-cut in a circuit $N = (V, E)$ can be found in $O(|V||E|)$ time.*

Min-Cut Balanced-Bipartition Heuristic

First, a repeated max-flow min-cut heuristic algorithm, flow-balanced bipartition (FBB), is developed for finding an r-balanced bipartition that minimizes the number of crossing nets. Then, an efficient implementation of FBB is developed that has the same asymptotic time complexity as one max-flow computation. For ease of presentation, the FBB algorithm is described on the original circuit rather than the flow network constructed from the circuit. The heuristic algorithm is described in Fig. 1. Figure 6 shows an example.

Table 2 compares the best bipartition net-cut sizes of FBB with those produced by the analytical-method-based partitioners EIG1 (Hagen and Kahng [7]) and PARABOLI (PB) (Riess et al. [13]). The results produced by PARABOLI were the best previously known results reported on the benchmark circuits. The results for FBB were the best of ten runs. On average, FBB outperformed EIG1 and PARABOLI by 58.1% and 11.3% respectively. For circuit S38417, the suboptimal result from FBB can be improved by (1) running more times and (2) applying clustering techniques to the circuit based on connectivity before partitioning.

In the FBB algorithm, the node-collapsing method is chosen instead of a more gradual method (e. g., [9]) to ensure that the capacity of a cut always reflects the real net-cut size. To pick a node at steps 4.2 and 5.2, a threshold R is given for the number of nodes in the uncollapsed subcircuit. A node is randomly picked if the number of nodes is larger than R. Otherwise, all nodes adjacent to C are tried and the one whose collapse induces a min-net-cut with the smallest size is picked. A naive implementation of step 2 by computing the max-flow from the zero flow would incur a high time complexity. Instead, the flow value in the flow network is retained, and additional flow is explored to saturate the bridging edges of the min-net-cut from one iteration to the next. The procedure is shown in Fig. 2. Initially, the flow network retains the flow function computed in the previous iteration. Since the max-flow computation using the augmenting-path method is insensitive to the initial flow values in the flow network and the order in which the augmenting paths are found, the above procedure correctly finds a max-flow with the same flow value as a max-flow computed in the collapsed flow network from the zero flow.

Theorem 2 *FBB has time complexity $O(|V||E|)$ for a connected circuit $N = (V, E)$.*

Theorem 3 *The number of iterations and the final net-cut size are nonincreasing functions of ϵ.*

In practice, FBB terminates much faster than this worst-case time complexity as shown in the Sect. "Experimental Results". Theorem 3 allows us to improve the efficiency of FBB and the partition quality for a larger ϵ. This is not true for other partitioning approaches such as the K&L heuristics.

Applications

Circuit partitioning is a fundamental problem in many areas of VLSI layout and design automation. The FBB algorithm provides the first efficient predictable solution to the min-cut balanced-circuit-partitioning problem. It directly relates the efficiency and the quality of the solution produced by the algorithm to the deviation factor ϵ. The algorithm can be easily extended to handle nets with different weights by simply assigning the weight of a net to its bridging edge in the flow network. K-way min-cut partitioning for $K > 2$ can be accomplished by recursively applying FBB or by setting $r = 1/K$ and then using FBB to find one partition at a time. A flow-based method for directly solving the problem can be found in [12]. Prepartitioning circuit clustering according to the connectivity or the timing information of the circuit can be easily incorporated into FBB by treating a cluster as a node. Heuristic solutions based on K&L heuristics or simulated annealing with low temperature can be used to further fine-tune the solution.

Experimental Results

The FBB algorithm was implemented in SIS/MISII [1] and tested on a set of large ISCAS and MCNC benchmark circuits on a SPARC 10 workstation with 36-MHz CPU and 32 MB memory.

Table 1 compares the average bipartition results of FBB with those reported by Dasdan and Aykanat in [3]. SN is based on the K&L heuristic algorithm in Sanchis [14]. PFM3 is based on the K&L heuristic with free moves as described in [3]. For each circuit, SN was run 20 times and PFM3 10 times from different randomly generated initial partitions. FBB was run 10 times from different randomly selected s and t. With only one exception, FBB outperformed both SN and PFM3 on the five circuits. On average, FBB found a bipartition with 24.5% and 19.0% fewer crossing nets than SN and PFM3 respectively. The runtimes of SN, PFM3, and FBB were not compared since they were run on different workstations.

Cross References

- ► Approximate Maximum Flow Construction
- ► Circuit Placement
- ► Circuit Retiming
- ► Max Cut
- ► Minimum Bisection
- ► Multiway Cut
- ► Separators in Graphs

Recommended Reading

1. Brayton, R.K., Rudell, R., Sangiovanni-Vincentelli, A.L.: MIS: A Multiple-Level Logic Optimization. IEEE Trans. CAD **6**(6), 1061–1081 (1987)
2. Cong, J., Hagen, L., Kahng, A.: Net Partitions Yield Better Module Partitions. In: Proc. 29th ACM/IEEE Design Automation Conf., 1992, pp. 47–52
3. Dasdan, A., Aykanat, C.: Improved Multiple-Way Circuit Partitioning Algorithms. In: Int. ACM/SIGDA Workshop on Field Programmable Gate Arrays, Feb. 1994
4. Fiduccia, C.M., Mattheyses, R.M.: A Linear Time Heuristic for Improving Network Partitions. In: Proc. ACM/IEEE Design Automation Conf., 1982, pp. 175–181
5. Garey, M., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, Gordonsville (1979)
6. Goldberg, A.W., Tarjan, R.E.: A New Approach to the Maximum Flow Problem. J. SIAM **35**, 921–940 (1988)

7. Hagen, L., Kahng, A.B.: Fast Spectral Methods for Ratio Cut Partitioning and Clustering. In: Proc. IEEE Int. Conf. on Computer-Aided Design, November 1991, pp. 10–13
8. Hu, T.C., Moerder, K.: Multiterminal Flows in a Hypergraph. In: Hu, T.C., Kuh, E.S. (eds.) VLSI Circuit Layout: Theory and Design, pp. 87–93. IEEE Press (1985)
9. Iman, S., Pedram, M., Fabian, C., Cong, J.: Finding Uni-Directional Cuts Based on Physical Partitioning and Logic Restructuring. In: 4th ACM/SIGDA Physical Design Workshop, April 1993
10. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by Simulated Annealing. Science **4598**, 671–680 (1983)
11. Kernighan, B., Lin, S.: An Efficient Heuristic Procedure for Partitioning of Electrical Circuits. Bell Syst. Tech. J., 291–307 (1970)
12. Liu, H., Wong, D.F.: Network-Flow-based Multiway Partitioning with Area and Pin Constraints. IEEE Trans. CAD Integr. Circuits Syst. **17**(1), 50–59 (1998)
13. Riess, B.M., Doll, K., Frank, M.J.: Partitioning Very Large Circuits Using Analytical Placement Techniques. In: Proc. 31th ACM/IEEE Design Automation Conf., 1994, pp. 646–651
14. Sanchis, L.A.: Multiway Network Partitioning. IEEE Trans. Comput. **38**(1), 62–81 (1989)
15. Wei, Y.C., Cheng, C.K.: Towards Efficient Hierarchical Designs by Ratio Cut Partitioning. In: Proc. IEEE Int. Conf. on Computer-Aided Design, November 1989, pp. 298–301
16. Yang, H., Wong, D.F.: Efficient Network Flow Based Min-Cut Balanced Partitioning. In: Proc. IEEE Int. Conf. on Computer-Aided Design, 1994, pp. 50–55

Circuit Placement

2000; Caldwell, Kahng, Markov
2002; Kennings, Markov
2006; Kennings, Vorwerk

ANDREW A. KENNINGS[1], IGOR L. MARKOV[2]
[1] Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON, Canada
[2] Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI, USA

Keywords and Synonyms

EDA; Netlist; Layout; Min-cut placement; Min-cost max-flow; Analytical placement; Mathematical programming

Problem Definition

This problem is concerned with efficiently determining constrained positions of objects while minimizing a measure of interconnect between the objects, as in physical layout of integrated circuits, commonly done in 2-dimensions. While most formulations are NP-hard, modern circuits are so large that practical algorithms for placement must have near-linear runtime and memory requirements, but not necessarily produce optimal solutions. While early software for circuit placement was based on Simulated Annealing, research in algorithms identified more scalable techniques which are now being adopted in the Electronic Design Automation industry.

One models a circuit by a hypergraph $G_h(V_h,E_h)$ with (i) vertices $V_h = \{v_1,\dots,v_n\}$ representing logic gates, standard cells, larger modules, or fixed I/O pads and (ii) hyperedges $E_h = \{e_1,\dots,e_m\}$ representing connections between modules. Every incident pair of a vertex and a hyperedge connect through a *pin* for a total of P pins in the hypergraph. Each vertex $v_i \in V_h$ has width w_i, height h_i and area A_i. Hyperedges may also be weighted. Given G_h, circuit placement seeks center positions (x_i,y_i) for vertices that optimize a hypergraph-based objective subject to constraints (see below). A placement is captured by $\mathbf{x} = (x_1,\cdots,x_n)$ and $\mathbf{y} = (y_1,\cdots,y_n)$.

Objective Let C_k be the index set of the hypergraph vertices incident to hyperedge e_k. The total half-perimeter wirelength (HPWL) of the circuit hypergraph is given by $\text{HPWL}(G_h) = \sum_{e_k\in E_h} \text{HPWL}(e_k) = \sum_{e_k\in E_h} \left[\max_{i,j\in C_k} |x_i - x_j| + \max_{i,j\in C_k} |y_i - y_j|\right]$. HPWL is piece-wise linear, separable in the x and y directions, convex, but not strictly convex. Among many objectives for circuit placement, it is the simplest and most common.

Constraints

1. **No overlap.** The area occupied by any two vertices cannot overlap; i. e., either $|x_i - x_j| \geq \frac{1}{2}(w_i + w_j)$ or $|y_i - y_j| \geq \frac{1}{2}(h_i + h_j), \forall v_i, v_j \in V_h$.
2. **Fixed outline.** Each vertex $v_i \in V_h$ must be placed entirely within a specified rectangular region bounded by $x_{\min}(y_{\min})$ and $x_{\max}(y_{\max})$ which denote the left (bottom) and right (top) boundaries of the specified region.
3. **Discrete slots.** There is only a finite number of discrete positions, typically on a grid. However, in large-scale circuit layout, slot constraints are often ignored during *global placement*, and enforced only during *legalization* and *detail placement*.

Other constraints may include alignment, minimum and maximum spacing, etc. Many placement techniques temporarily relax overlap constraints into density constraints to avoid vertices clustered in small regions. A $m \times n$ regular bin structure B is superimposed over the fixed outline and vertex area is assigned to bins based on the positions of vertices. Let D_{ij} denote the density of bin $B_{ij} \in B$, defined as the total cell area assigned to bin B_{ij} divided by its capacity. Vertex overlap is limited implicitly by $D_{ij} \leq K, \forall B_{ij} \in B$, for some $K \leq 1$ (density target).

Problem 1 (Circuit Placement)
INPUT: *Circuit hypergraph $G_h(V_h,E_h)$ and a fixed outline for the placement area.*
OUTPUT: *Positions for each vertex $v_i \in V_h$ such that (1) wirelength is minimized and (2) the area-density constraints $D_{ij} \leq K$ are satisfied for all $B_{ij} \in B$.*

Key Results

An unconstrained optimal position of a single placeable vertex connected to fixed vertices can be found in linear time as the median of adjacent positions [8]. Unconstrained HPWL minimization for multiple placeable vertices can be formulated as a linear program [7,10]. For each $e_k \in E_h$, upper and lower bound variables U_k and L_k are added. The cost of e_k (x-direction only) is the difference between U_k and L_k. Each $U_k(L_k)$ comes with p_k inequality constraints that restricts its value to be larger (smaller) than the position of every vertex $i \in C_k$. A hypergraph with n vertices and m hyperedges is represented by a linear program with $n + 2m$ variables and $2P$ constraints.

Linear programming has poor scalability, and integrating constraint-tracking into optimization is difficult. Other approaches include non-linear optimization and partitioning-based methods.

Combinatorial Techniques for Wirelength Minimization

The no-overlap constraints are not convex and cannot be directly added to the linear program for HPWL minimization. Such a program is first solved directly or by casting its dual as an instance of the min-cost max-flow problem [12]. Vertices often cluster in small regions of high density. One can lower-bound the distance between closely-placed vertices with a single linear constraint that depends on the relative placement of these vertices [10]. The resulting optimization problem is incrementally re-solved, and the process repeats until the desired density is achieved.

The *min-cut placement* technique is based on balanced min-cut partitioning of hypergraphs and is more focused on density constraints [11]. Vertices of the initial hypergraph are first partitioned in two similar-sized groups. One of them is assigned to the left half of the placement region, and the other one to the right half. Partitioning is performed by the Multi-level Fiduccia–Mattheyses (MLFM) heuristic [9] to minimize connections between the two groups of vertices (the net-cut objective). Each half is partitioned again, but takes into account the connections to the other half [11]. At the large scale, ensuring the similar sizes of bi-partitions corresponds to density constraints and cut minimization corresponds to HPWL minimization. When regions become small and contain $<$ 10 vertices, optimal positions can be found with respect to discrete slot constraints by branch-and-bound [2]. Balanced hypergaph partitioning is NP-hard [4], but the MLFM heuristic takes $O((V + E) \log V)$ time. The entire min-cut placement procedure takes $O((V + E)(\log V)^2)$ time and can process hypergraphs with millions of vertices in several hours.

A special case of interest is that of one-dimensional placement. When all vertices have identical width and none of them are fixed, one obtains the NP-hard MINIMUM LINEAR ARRANGEMENT problem [4] which can be approximated in polynomial time within $O(\log V)$ and solved exactly for trees in $O(V^3)$ time as shown by Yannakakis. The min-cut technique described above also works well for the related NP-hard MINIMUM-CUT LINEAR ARRANGEMENT problem [4].

Nonlinear Optimization

Quadratic and generic non-linear optimization may be faster than linear programming, while reasonably approximating the original formulation. The hypergraph is represented by a weighted graph where w_{ij} represents the weight on the 2-pin edge connecting vertices v_i and v_j in the weighted graph. When an edge is absent, $w_{ij} = 0$, and in general $w_{ii} = -\Sigma_{i \neq j} w_{ij}$.

Quadratic Placement A quadratic placement (x-direction only) is given by

$$\Phi(x) = \sum_{i,j} w_{ij}\left[(x_i - x_j)^2\right] = \frac{1}{2}\mathbf{x}^{\mathrm{T}}\mathbf{Q}\mathbf{x} + \mathbf{c}^{\mathrm{T}}\mathbf{x} + \text{const.} \quad (1)$$

The global minimum of $\Phi(x)$ is found by solving $\mathbf{Qx}+\mathbf{c} = \mathbf{0}$ which is a sparse, symmetric, positive-definite system of linear equations (assuming $\geq$ 1 fixed vertex), efficiently solved to sufficient accuracy using any number of iterative solvers. Quadratic placement may have different optima depending on the model (clique or star) used to represent hyperedges. However, for a k-pin hyperedge, if the weight on the 2-pin edges introduced is set to W_c in the clique mode and kW_c in the star model, then the models are equivalent in quadratic placement [7].

Linearized Quadratic Placement Quadratic placement can produce lower quality placements. To approximate the linear objective, one can iteratively solve Eq. (1) with $w_{ij} = 1/|x_i - x_j|$ computed at every iteration. Alternatively, one can solve a single β-regularized optimization problem given by $\Phi^{\beta}(\mathbf{x}) = \min_x \sum_{i,j} w_{ij}\sqrt{(x_i - x_j)^2 + \beta}$, $\beta > 0$,

e. g., using a Primal-Dual Newton method with quadratic convergence [1].

Half-Perimeter Wirelength Placement HPWL can be provably approximated by strictly convex and differentiable functions. For 2-pin hyperedges, β-regularization can be used [1]. For an m-pin hyperedge ($m \geq 3$), one can rewrite HPWL as the maximum (l_∞-norm) of all $m(m-1)/2$ pairwise distances $|x_i - x_j|$ and approximate the l_∞-norm by the l_p-norm (p-th root of the sum of p-th powers). This removes all non-differentiabilities except at **0** which is then removed with β-regularization. The resulting HPWL approximation is given by

$$\mathrm{HPWL}_{p-\beta-\mathrm{reg}}(G_h) = \sum_{e_k \in E_h} \Big(\sum_{i,j \in C_k} |x_i - x_j|^p + \beta \Big)^{1/p} \tag{2}$$

which overestimates HPWL with arbitrarily small relative error as $p \to \infty$ and $\beta \to 0$ [7]. Alternatively, HPWL can be approximated via the log-sum-exp formula given by

$$\mathrm{HPWL}_{\text{log-sum-exp}}(G_h) = \alpha \sum_{e_k \in E_h} \left[\ln \Big(\sum_{i \in C_k} \exp \Big(\frac{x_i}{\alpha} \Big) \Big) + \ln \Big(\sum_{v_i \in C_k} \exp \Big(\frac{-x_i}{\alpha} \Big) \Big) \right] \tag{3}$$

where $\alpha > 0$ is a smoothing parameter [6]. Both approximations can be optimized using conjugate gradient methods.

Analytic Techniques for Target Density Constraints

The target density constraints are non-differentiable and are typically handled by approximation.

Force-Based Spreading The key idea is to add constant forces **f** that pull vertices always from overlaps, and recompute the forces over multiple iterations to reflect changes in vertex distribution. For quadratic placement, the new optimality conditions are $\mathbf{Qx} + \mathbf{c} + \mathbf{f} = \mathbf{0}$ [8]. The constant force can perturb a placement in any number of ways to satisfy the target density constraints. The force **f** is computed using a discrete version of Poisson's equation.

Fixed-Point Spreading A fixed point f is a pseudo-vertex with zero area, fixed at (x_f, y_f), and connected to one vertex $H(f)$ in the hypergraph through the use of a pseudo-edge with weight $w_{f,H(f)}$. Quadratic placement with fixed points is given by $\Phi(x) = \sum_{i,j} w_{i,j}(x_i - x_j)^2 + \sum_f w_{f,H(f)}(x_{H(f)} - x_f)^2$. Each each fixed point f introduces a quadratic term $w_{f,H(f)}(x_{H(f)} - x_f)^2$. By manipulating the positions of fixed points, one can perturb a placement to satisfy the target density constraints. Compared to constant forces, fixed points improve the controllability and stability of placement iterations [5].

Generalized Force-Directed Spreading The Helmholtz equation models a diffusion process and makes it ideal for spreading vertices [3]. The Helmholtz equation is given by

$$\frac{\partial^2 \phi(x,y)}{\partial x^2} + \frac{\partial^2 \phi(x,y)}{\partial y^2} - \epsilon \phi(x,y) = D(x,y)\,,$$
$$(x,y) \in R \quad \frac{\partial \phi}{\partial v} = 0\,,$$
$$(x,y) \text{ on the boundary of } R \tag{4}$$

where $\epsilon > 0$, v is an outer unit normal, R represents the fixed outline, and $D(x,y)$ represents the continuous density function. The boundary conditions, $\partial\phi/\partial v = 0$, specify that forces pointing outside of the fixed outline be set to zero – this is a key difference with the Poisson method which assumes that forces become zero at infinity. The value ϕ_{ij} at the center of each bin B_{ij} is found by discretization of Eq. (4) using finite differences. The density constraints are replaced by $\phi_{ij} = \hat{K}, \forall B_{ij} \in B$ where $\hat{K}$ is a scaled representative of the density target K. Wirelength minimization subject to the smoothed density constraints can be solved via Uzawa's algorithm. For quadratic wirelength, this algorithm is a generalization of force-based spreading.

Potential Function Spreading Target density constraints can also be satisfied via a penalty function. The area assigned to bin B_{ij} by vertex v_i is represented by $\mathrm{Potential}(v_i, B_{ij})$ which is a bell-shaped function. The use of piecewise quadratic functions make the potential function non-convex, but smooth and differentiable [6]. The penalty term given by

$$\mathrm{Penalty} = \sum_{B_{ij} \in B} \Big(\sum_{v_i \in V_h} \mathrm{Potential}(v_i, B_{ij}) - K \Big)^2 \tag{5}$$

can be combined with a wirelength approximation to arrive at an unconstrained optimization problem which is solved using an efficient conjugate gradient method [6].

Applications

Practical applications involve more sophisticated interconnect objectives, such as circuit delay, routing congestion, power dissipation, power density, and maximum thermal gradient. The above techniques are adapted to handle multi-objective optimization. Many such extensions are based on heuristic assignment of net weights that encourage the shortening of some (e. g., timing-critical and frequently-switching) connections at the expense of other connections. To moderate routing congestion, predictive congestion maps are used to decrease the maximal density constraint for placement in congested regions. Another application is in physical synthesis, where incremental placement is used to evaluate changes in circuit topology.

Experimental Results

Circuit placement has been actively studied for the past 30 years and a wealth of experimental results are reported throughout the literature. A 2003 result demonstrated that placement tools could produce results as much as 1.41× to 2.09× known optimal wirelengths on average (advances have been made since this study). A 2005 placement contest found that a set of tools produced placements with wirelengths that differed by as much as 1.84× on average. A 2006 placement contest found that a set of tools produced placements that differed by as much as 1.39× on average when the objective was the simultaneous minimization of wirelength, routability and run time. Placement run times range from minutes for smaller instances to hours for larger instances, with several millions of variables.

Data Sets

Benchmarks include the ICCAD '04 suite (http://vlsicad.eecs.umich.edu/BK/ICCAD04bench/), the ISPD '05 suite (http://www.sigda.org/ispd2005/contest.htm) and the ISPD '06 suite (http://www.sigda.org/ispd2006/contest.htm). Instances in these benchmark suites contain between 10K to 2.5M placeable objects. Other common suites can be found, including large-scale placement instances problems with known optimal solutions (http://cadlab.cs.ucla.edu/~pubbench).

Cross References

► Performance-Driven Clustering

Recommended Reading

1. Alpert, C.J., Chan, T., Kahng, A.B., Markov, I.L., Mulet, P.: Faster minimization of linear wirelength for global placement. IEEE Trans. CAD **17**(1), 3–13 (1998)
2. Caldwell, A.E., Kahng, A.B., Markov, I.L.: Optimal partitioners and end-case placers for standard-cell layout. IEEE Trans. CAD **19**(11), 1304–1314 (2000)
3. Chan, T., Cong, J., Sze, K.: Multilevel generalized force-directed method for circuit placement. Proc. Intl. Symp. Physical Design. ACM Press, San Francisco, 3–5 Apr 2005. pp. 185–192 (2005)
4. Ausiello, G., Crescenzi, P., Gambosi, G., Kann, V., Marchetti-Spaccamela, A., Protasi, M.: Complexity and Approximation: Combinatorial optimization problems and their approximability properties. Springer (1998)
5. Hu, B., Marek-Sadowska, M.: Multilevel fixed-point-addition-based VLSI placement. IEEE Trans. CAD **24**(8), 1188–1203 (2005)
6. Kahng, A.B., Wang, Q.: Implementation and extensibility of an analytic placer. IEEE Trans. CAD **24**(5), 734–747 (2005)
7. Kennings, A., Markov, I.L.: Smoothing max-terms and analytical minimization of half-perimeter wirelength. VLSI Design **14**(3), 229–237 (2002)
8. Kennings, A., Vorwerk, K.: Force-directed methods for generic placement. IEEE Trans. CAD **25**(10), 2076–2087 (2006)
9. Papa, D.A., Markov, I.L.: Hypergraph partitioning and clustering. In: Gonzalez, T. (ed.) Handbook of algorithms. Taylor & Francis Group, Boca Raton, CRC Press, pp. 61–1 (2007)
10. Reda, S., Chowdhary, A.: Effective linear programming based placement methods. In: ACM Press, San Jose, 9–12 Apr 2006
11. Roy, J.A., Adya, S.N., Papa, D.A., Markov, I.L.: Min-cut floorplacement. IEEE Trans. CAD **25**(7), 1313–1326 (2006)
12. Tang, X., Tian, R., Wong, M.D.F.: Optimal redistribution of white space for wirelength minimization. In: Tang, T.-A. (ed.) Proc. Asia South Pac. Design Autom. Conf., ACM Press, 18–21 Jan 2005, Shanghai. pp. 412–417 (2005)

Circuit Retiming

1991; Leiserson, Saxe

HAI ZHOU
Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL, USA

Keywords and Synonyms

Min-period retiming; Min-area retiming

Problem Definition

Circuit retiming is one of the most effective structural optimization techniques for sequential circuits. It moves the registers within a circuit without changing its function. Besides clock period, retiming can be used to minimize the number of registers in the circuit. It is also called minimum area retiming problem. Leiserson and Saxe [3] started the research on retiming and proposed algorithms for both minimum period and minimum area retiming. Both their algorithms for minimum area and minimum period will be presented here.

The problems can be formally described as follows. Given a directed graph $G = (V, E)$ representing a circuit—each node $v \in V$ represents a gate and each edge $e \in E$ represents a signal passing from one gate to another—with gate delays $d\colon V \to \mathbb{R}^+$ and register numbers $w\colon E \to \mathbb{N}$, the minimum area problem asks for a relocation of registers $w'\colon E \to \mathbb{N}$ such that the number of registers in the circuit is minimum under a given clock period φ. The minimum period problem asks for a solution with the minimum clock period.

Notations

To guarantee that the new registers are actually a relocation of the old ones, a label $r\colon V \to \mathbb{Z}$ is used to represent how many registers are moved from the outgoing edges to the incoming edges of each node. Using this notation, the new number of registers on an edge (u, v) can be computed as

$$w'[u,v] = w[u,v] + r[v] - r[u]\,.$$

The same notation can be extended from edges to paths. However, between any two nodes u and v, there may be more than one path. Among these paths, the ones with the minimum number of registers will decide how many registers can be moved outside of u and v. The number is denoted by $W[u,v]$ for any $u, v \in V$, that is,

$$W[u,v] \triangleq \min_{p\colon u \rightsquigarrow v} \sum_{(x,y)\in p} w[x,y]$$

The maximal delay among all the paths from u to v with the minimum number of registers is also denoted by $D[u,v]$, that is,

$$D[u,v] \triangleq \max_{w[p\colon u \rightsquigarrow v]=W[u,v]} \sum_{x\in p} d[x]$$

Constraints

Based on the notations, a valid retiming r should not have any negative number of registers on any edge. Such a validity condition is given as

$$P0(r) \triangleq \forall (u,v) \in E\colon w[u,v] + r[v] - r[u] \geq 0$$

On the other hand, given a retiming r, the minimum number of registers between any two nodes u and v is $W[u,v] - r[u] + r[v]$. This number will not be negative because of the previous constraint. However, when it is zero, there will be a path of delay $D[u,v]$ without any register on it. Therefore, to have a retimed circuit working for clock period φ, the following constraint must be satisfied.

$$P1(r) \triangleq \forall u, v \in V\colon D[u,v] > \phi \\ \Rightarrow W[u,v] + r[v] - r[u] \geq 1$$

Key Results

The object of the minimum area retiming is to minimize the total number of registers in the circuit, which is given by $\sum_{(u,v)\in E} w'[u,v]$. Expressing $w'[u,v]$ in terms of r, the objective becomes

$$\sum_{v\in V} (in[v] - out[v]) * r[v] + \sum_{(u,v)\in E} w[u,v]$$

where $in[v]$ is the in-degree and $out[v]$ is the out-degree of node v. Since the second term is a constant, the problem can be formulated as the following integer linear program.

$$\begin{aligned} &\text{Minimize} \sum_{v\in V} (in[v] - out[v]) * r[v] \\ &s.t.\ \ w[u,v] + r[v] - r[u] \geq 0 \quad \forall (u,v) \in E \\ &\qquad W[u,v] + r[v] - r[u] \geq 1 \ \ \forall u, v \in V\colon D[u,v] > \phi \\ &\qquad r[v] \in \mathbb{Z} \quad \forall v \in V \end{aligned}$$

Since the constraints have only difference inequalities with integer constant terms, solving the relaxed linear program (without the integer constraint) will only give integer solutions. Even better, it can be shown that the problem is the dual of a minimum cost network flow problem, and thus can be solved efficiently.

Theorem 1 *The integer linear program for the minimum area retiming problem is the dual of the following minimum cost network flow problem.*

$$\begin{aligned} &\mathit{Minimize} \sum_{(u,v)\in E} w[u,v] * f[u,v] \\ &\qquad + \sum_{D[u,v]>\phi} (W[u,v] - 1) * f[u,v] \\ &s.t.\ \ in[v] + \sum_{(v,w)\in E \vee D[v,w]>\phi} f[v,w] = out[v] \\ &\qquad + \sum_{(u,v)\in E D[u,v]>\phi} f[u,v] \quad \forall v \in V \\ &\qquad f[u,v] \geq 0 \quad \forall (u,v) \in E D[u,v] > \phi \end{aligned}$$

From the theorem, it can be seen that the network graph is a dense graph where a new edge (u,v) needs to be introduced for any node pair u, v such that $D[u,v] > \phi$.

There may be redundant constraints in the system. For example, if $W[u,w] = W[u,v] + w[v,w]$ and $D[u,v] > \phi$ then the constraint $W[u,w] + r[w] - r[u] \geq 1$ is redundant, since there are already $W[u,v] + r[v] - r[u] \geq 1$ and $w[v,w] + r[w] - r[v] \geq 0$. However, it may not be easy to check and remove all redundancy in the constraints.

In order to build the minimum cost flow network, it is needed to first compute both matrices W and D. Since $W[u,v]$ is the shortest path from u to v in terms of w, the computation of W can be done by an all-pair shortest paths algorithm such as Floyd–Warshall's algorithm [1]. Furthermore, if the ordered pair $(w[x,y], -d[x])$ is used as the edge weight for each $(x,y) \in E$, an all-pair shortest paths algorithm can also be used to compute both W and D. The algorithm will add weights by component-wise addition and will compare weights by lexicographic ordering.

Leiserson and Saxe [3]'s first algorithm for the minimum period retiming was also based on the matrices W and D. The idea was that the constraints in the integer linear program for the minimum area retiming can be checked efficiently by Bellman–Ford's shortest paths algorithm [1], since they are just difference inequalities. This gives a feasibility checking for any given clock period φ. Then the optimal clock period can be found by a binary search on a range of possible periods. The feasibility checking can be done in $O(|V|^3)$ time, thus the runtime of such an algorithm is $O(|V|^3 \log |V|)$.

Their second algorithm got rid of the construction of the matrices W and D. It still used a clock period feasibility checking within a binary search. However, the feasibility checking was done by incremental retiming. It works as follows. Starting with $r = 0$, the algorithm computes the arrival time of each node by the longest paths computation on a DAG (Directed Acyclic Graph). For each node v with an arrival time larger than the given period φ, the $r[v]$ will be increased by one. The process of the arrival time computation and r increasing will be repeated $|V| - 1$ times. After that, if there is still arrival time that is larger than φ, then the period is infeasible. Since the feasibility checking is done in $O(|V||E|)$ time, the runtime for the minimum period retiming is $O(|V||E| \log |V|)$.

Applications

Shenoy and Rudell [7] implemented Leiserson and Saxe's minimum period and minimum area retiming algorithms with some efficiency improvements. For minimum period retiming, they implemented the second algorithm and, in order to find out infeasibility earlier, they introduced a pointer from one node to another where at least one register is required between them. A cycle formed by the pointers indicates the infeasibility of the given period. For minimum area retiming, they removed some of the redundancy in the constraints and used the cost-scaling algorithm of Goldberg and Tarjan [2] for the minimum cost flow computation.

Open Problems

As can be seen from the second minimum period retiming algorithm here and Zhou's algorithm [8] in another entry (► Circuit Retiming: An Incremental Approach), incremental computation of the longest combinational paths (i. e. those without register on them) is more efficient than constructing the dense graph (via matrices W and D). However, the minimum area retiming algorithm is still based on a minimum cost network flow on the dense graph. An interesting open question is to see whether a more efficient algorithm based on incremental retiming can be designed for the minimum area problem.

Experimental Results

Sapatnekar and Deokar [6] and Pan [5] proposed continuous retiming as an efficient approximation for minimum period retiming, and reported the experimental results. Maheshwari and Sapatnekar [4] also proposed some efficiency improvements to the minimum area retiming algorithm and reported their experimental results.

Cross References

► Circuit Retiming: An Incremental Approach

Recommended Reading

1. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 2nd edn. MIT Press, Cambridge (2001)
2. Goldberg, A.V., Tarjan, R.E.: Solving minimum cost flow problem by successive approximation. In: Proc. ACM Symposium on the Theory of Computing, pp. 7–18 (1987). Full paper in: Math. Oper. Res. **15**, 430–466 (1990)
3. Leiserson, C.E., Saxe, J.B.: Retiming synchronous circuitry. Algorithmica **6**, 5–35 (1991)
4. Maheshwari, N., Sapatnekar, S.S.: Efficient retiming of large circuits, IEEE Transactions on Very Large-Scale Integrated Systems. **6**, 74–83 (1998)
5. Pan, P.: Continuous retiming: Algorithms and applications. In: Proc. Intl. Conf. Comput. Design, pp. 116–121. IEEE Press, Los Almitos (1997)
6. Sapatnekar, S.S., Deokar, R.B.: Utilizing the retiming-skew equivalence in a practical algorithm for retiming large circuits. IEEE Trans. Comput. Aided Des. **15**, 1237–1248 (1996)
7. Shenoy, N., Rudell, R.: Efficient implementation of retiming. In Proc. Intl. Conf. Computer-Aided Design, pp. 226–233. IEEE Press, Los Almitos (1994)

8. Zhou, H.: Deriving a new efficient algorithm for min-period retiming. In Asia and South Pacific Design Automation Conference, Shanghai, China, Jan. ACM Press, New York (2005)

Circuit Retiming: An Incremental Approach

2005; Zhou

HAI ZHOU
Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL, USA

Keywords and Synonyms

Minimum period retiming; Min-period retiming

Problem Definition

Circuit retiming is one of the most effective structural optimization techniques for sequential circuits. It moves the registers within a circuit without changing its function. The minimal period retiming problem needs to minimize the longest delay between any two consecutive registers, which decides the clock period.

The problem can be formally described as follows. Given a directed graph $G = (V, E)$ representing a circuit – each node $v \in V$ represents a gate and each edge $e \in E$ represents a signal passing from one gate to another – with gate delays $d\colon V \to \mathbb{R}^+$ and register numbers $w\colon E \to \mathbb{N}$, it asks for a relocation of registers $w'\colon E \to \mathbb{N}$ such that the maximal delay between two consecutive registers is minimized.

Notations To guarantee that the new registers are actually a relocation of the old ones, a label $r\colon V \to \mathbb{Z}$ is used to represent how many registers are moved from the outgoing edges to the incoming edges of each node. Using this notation, the new number of registers on an edge (u,v) can be computed as

$$w'[u,v] = w[u,v] + r[v] - r[u] .$$

Furthermore, to avoid explicitly enumerating the paths in finding the longest path, another label $t\colon V \to \mathbb{R}^+$ is introduced to represent the output arrival time of each gate, that is, the maximal delay of a gate from any preceding register. The condition for t to be at least the combinational delays is

$$\forall (u,v) \in E\colon w'[u,v] = 0 \Rightarrow t[v] \geq t[u] + d[v] .$$

Constraints and Objective Based on the notations, a valid retiming r should not have any negative number of registers on any edge. Such a validity condition is given as

$$P0(r) \triangleq \forall (u,v) \in E\colon w[u,v] + r[v] - r[u] \geq 0 .$$

As already stated, the conditions for t to be valid arrival time is given by the following two predicates.

$$P1(t) \triangleq \forall v \in V\colon t[v] \geq d[v]$$
$$P2(r,t) \triangleq \forall (u,v) \in E\colon r[u] - r[v] = w[u,v] \Rightarrow t[v] - t[u] \geq d[v] .$$

A predicate P is used to denote the conjunction of the above conditions:

$$P(r,t) \triangleq P0(r) \wedge P1(t) \wedge P2(r,t) .$$

A minimal period retiming is a solution $\langle r, t\rangle$ satisfying the following optimality condition:

$$P3 \triangleq \forall r', t'\colon P(r',t') \Rightarrow \max(t) \leq \max(t') ,$$

where

$$\max(t) \triangleq \max_{v \in V} t[v] .$$

Since only a valid retiming (r', t') will be discussed in the sequel, to simplify the presentation, the range condition $P(r', t')$ will often be omitted; the meaning shall be clear from the context.

Key Results

This section will show how an efficient algorithm is designed for the minimal period retiming problem. Contrary to the usual way of only presenting the final product, i. e. the algorithm, but not the ideas on its design, a step-by-step design process will be shown to finally arrive at the algorithm.

To design an algorithm is to construct a procedure such that it will terminate in finite steps and will satisfy a given predicate when it terminates. In the minimal period retiming problem, the predicate to be satisfied is $P0 \wedge P1 \wedge P2 \wedge P3$. The predicate is also called the *post-condition*. It can be argued that any non-trivial algorithm will have at least one loop, otherwise, the processing length is only proportional to the text length. Therefore, some part of the post-condition will be iteratively satisfied by the loop, while the remaining part will be initially satisfied by an initialization and made invariant during the loop.

The first decision needed to make is to partition the post-condition into possible invariant and loop goal. Among the four conjuncts, the predicate $P3$ gives the optimality condition and is the most complex one. Therefore, it will be used as a loop goal. On the other hand, the predicates $P0$ and $P1$ can be easily satisfied by the following simple initialization.

$$r, t := 0, d\ .$$

Based on these, the plan is to design an algorithm with the following scheme.

```
$r, t := 0, d$
do$\{P0 \wedge P1\}$
    $\neg P2 \rightarrow$ update $t$
    $\neg P3 \rightarrow$ update $r$
od$\{P0 \wedge P1 \wedge P2 \wedge P3\}$ .
```

The first command in the loop can be refined as

$$\exists (u, v) \in E\colon r[u] - r[v] = w[u, v] \wedge t[v] - t[u] < d[v] \rightarrow t[v] := t[u] + d[v]\ .$$

This is simply the Bellman–Ford relaxations for computing the longest paths.

The second command is more difficult to refine. If $\neg P3$, that is, there exists another valid retiming $\langle r', t' \rangle$ such that $\max(t) > \max(t')$, then on any node v such that $t[v] = \max(t)$ it must have $t'[v] < t[v]$. One property known on these nodes is

$$\forall v \in V\colon t'[v] < t[v] \Rightarrow (\exists u \in V\colon r[u] - r[v] > r'[u] - r'[v])\ ,$$

which means that if the arrival time of v is smaller in another retiming $\langle r', t' \rangle$, then there must be a node u such that r' gives more registers between u and v. In fact, one such a u is the starting node of the longest combinational path to v that gives the delay of $t[v]$.

To reduce the clock period, the variable r needs to be updated to make it closer to r'. It should be noted that it is not the absolute values of r but their differences that are relevant in the retiming. If $\langle r, t \rangle$ is a solution to a retiming problem, then $\langle r + c, t \rangle$, where $c \in \mathbb{Z}$ is an arbitrary constant, is also a solution. Therefore r can be made "closer" to r' by allocating more registers between u and v, that is, by either decreasing $r[u]$ or increasing $r[v]$. Notice that v can be easily identified by $t[v] = \max(t)$. No matter whether $r[v]$ or $r[u]$ is selected to change, the amount of change should be only one since r should not be over-adjusted. Thus, after the adjustment, it is still true that $r[v] - r[u] \leq r'[v] - r'[u]$, or equivalently $r[v] - r'[v] \leq r[u] - r'[u]$. Since v is easy to identify, $r[v]$ is selected to increase. The arrival time $t[v]$ can be immediately reduced to $d[v]$. This gives a refinement of the second commend:

$$\neg P3 \wedge P2 \wedge \exists v \in V\colon t[v] = \max(t) \rightarrow r[v], t[v] := r[v] + 1, d[v]\ .$$

Since registers are moved in the above operation, the predicate $P2$ may be violated. However, the first command will take care of it. That command will increase t on some nodes; some may even become larger than $\max(t)$ before the register move. The same reasoning using $\langle r', t' \rangle$ shows that their r values shall be increased, too. Therefore, to implement this As-Soon-As-Possible (ASAP) increase of r, a snapshot of $\max(t)$ needs to be taken when $P2$ is valid. Physically, such a snapshot records one feasible clock period ϕ, and can be implemented by adding one more command in the loop:

$$P2 \wedge \phi > \max(t) \rightarrow \phi := \max(t)\ .$$

However, such an ASAP operation may increase $r[u]$ even when $w[u, v] - r[u] + r[v] = 0$ for an edge (u,v). It means that $P0$ may no longer be an invariant. But moving $P0$ from invariant to loop goal will not cause a problem since one more command can be added in the loop to take care of it:

$$\exists (u, v) \in E\colon r[u] - r[v] > w[u, v] \rightarrow r[v] := r[u] - w[u, v]\ .$$

Putting all things together, the algorithm now has the following form.

```
$r, t, \phi := 0, d, \infty;$
do$\{P1\}$
    $\exists (u, v) \in E\colon r[u] - r[v] = w[u, v]$
        $\wedge\ t[v] - t[u] < d[v] \rightarrow t[v] := t[u] + d[v]$
    $\neg P3 \wedge \exists v \in V\colon t[v] \geq \phi$
        $\rightarrow r[v], t[v] := r[v] + 1, d[v]$
    $P0 \wedge P2 \wedge \phi > \max(t) \rightarrow \phi := \max(t)$
    $\exists (u, v) \in E\colon r[u] - r[v] > w[u, v]$
        $\rightarrow r[v] := r[u] - w[u, v]$
od$\{P0 \wedge P1 \wedge P2 \wedge P3\}$ .
```

The remaining task to complete the algorithm is how to check $\neg P3$. From previous discussion, it is already known that $\neg P3$ implies that there is a node u such that $r[u] - r'[u] \geq r[v] - r'[v]$ every time after $r[v]$ is increased. This means that $\max_{v \in V} r[v] - r'[v]$ will not increase. In

Circuit Retiming: An Incremental Approach, Table 1
Experimental Results

name	#gates	clock period		$\sum r$	#updates	time(s)	ASTRA	
		before	after				A(s)	B(s)
s1423	490	166	127	808	7619	0.02	0.03	0.02
s1494	558	89	88	628	7765	0.02	0.01	0.01
s9234	2027	89	81	2215	76943	0.12	0.11	0.09
s9234.1	2027	89	81	2164	77644	0.16	0.11	0.10
s13207	2573	143	82	4086	28395	0.12	0.38	0.12
s15850	3448	186	77	12038	99314	0.36	0.43	0.17
s35932	12204	109	100	16373	108459	0.28	0.24	0.65
s38417	8709	110	56	9834	155489	0.58	0.89	0.64
s38584	11448	191	163	19692	155637	0.41	0.50	0.67
s38584.1	11448	191	183	9416	114940	0.48	0.55	0.78

other words, there is at least one node v whose $r[v]$ will not change. Before $r[v]$ is increased, it also has $w_{u \rightsquigarrow v} - r[u] + r[v] \leq 0$, where $w_{u \rightsquigarrow v} \geq 0$ is the original number of registers on one path from u to v, which gives $r[v] - r[u] \leq 1$ even after the increase of $r[v]$. This implies that there will be at least $i + 1$ nodes whose r is at most i for $0 \leq i < |V|$. In other words, the algorithm can keep increasing r and when there is any r reaching $|V|$ it shows that $P3$ is satisfied. Therefore, the complete algorithm will have the following form.

```
$r, t, \phi := 0, d, \infty;$
do$\{P1\}$
    $\exists(u, v) \in E\colon r[u] - r[v] = w[u, v]$
        $\wedge\ t[v] - t[u] < d[v] \to t[v] := t[u] + d[v]$
    $(\forall v \in V\colon r[v] < |V|)$
        $\wedge\ \exists v \in V\colon t[v] \geq \phi \to r[v], t[v] := r[v] + 1, d[v]$
    $(\exists v \in V\colon r[v] \geq |V|)$
        $\wedge\ \exists v \in V\colon t[v] > \phi \to r[v], t[v] := r[v] + 1, d[v]$
    $P0 \wedge P2 \wedge \phi > \max(t) \to \phi := \max(t)$
    $\exists(u, v) \in E\colon r[u] - r[v] > w[u, v]$
        $\to r[v] := r[u] - w[u, v]$
od$\{P0 \wedge P1 \wedge P2 \wedge P3\}$ .
```

The correctness of the algorithm can be proved easily by showing that the invariant $P1$ is maintained and the negation of the guards implies $P0 \wedge P2 \wedge P3$. The termination is guaranteed by the monotonic increase of r and an upper bound on it. In fact, the following theorem gives its worst case runtime.

Theorem 1 *The worst case running time of the given retiming algorithm is upper bounded by $O(|V|^2|E|)$.*

The runtime bound of the retiming algorithm is got under the worst case assumption that each increase on r will trigger a timing propagation on the whole circuit ($|E|$ edges). This is only true when the r increase moves all registers in the circuit. However, in such a case, the r is upper bounded by 1, thus the running time is not larger than $O(|V||E|)$. On the other hand, when the r value is large, the circuit is partitioned by the registers into many small parts, thus the timing propagation triggered by one r increase is limited within a small tree.

Applications

In the basic algorithm, the optimality $P3$ is verified by an $r[v] \geq |V|$. However, in most cases, the optimality condition can be discovered much earlier. Since each time $r[v]$ is increased, there must be a "safe-guard" node u such that $r[u] - r'[u] \geq r[v] - r'[v]$ after the operation. Therefore, if a pointer is introduced from v to u when $r[v]$ is increased, the pointers cannot form a cycle under $\neg P3$. In fact, the pointers will form a forest where the roots have $r = 0$ and a child can have an r at most one larger than its parent. Using a cycle by the pointers as an indication of $P3$, instead of an $r[v] \geq |V|$, the algorithm can have much better practical performance.

Open Problems

Retiming is usually used to optimize either the clock period or the number of registers in the circuit. The discussed algorithm solves only the minimal period retiming problem. The retiming problem for minimizing the number of registers under a given period has been solved by Leiserson and Saxe [1] and is presented in another entry in this encyclopedia. Their algorithm reduces the problem to the dual of a minimal cost network problem on a denser graph. An interesting open question is to see whether an efficient iterative algorithm similar to Zhou's algorithm can be designed for the minimal register problem.

Experimental Results

Experimental results are reported by Zhou [3] which compared the runtime of the algorithm with an efficient heuristic called ASTRA [2]. The results on the ISCAS89 benchmarks are reproduced here in Table 1 from [3], where columns A and B are the running time of the two stages in ASTRA.

Cross References

► Circuit Retiming

Recommended Reading

1. Leiserson, C.E., Saxe, J.B.: Retiming synchronous circuitry. Algorithmica **6**, 5–35 (1991)
2. Sapatnekar, S.S., Deokar, R.B.: Utilizing the retiming-skew equivalence in a practical algorithm for retiming large circuits. IEEE Transactions on Computer Aided Design **15**, 1237–1248 (1996)
3. Zhou, H.: Deriving a new efficient algorithm for min-period retiming. In: Asia and South Pacific Design Automation Conference, Shanghai, China, January 2005

Clock Synchronization

1994; Patt-Shamir, Rajsbaum

Boaz Patt-Shamir
Department of Electrical Engineering,
Tel-Aviv University, Tel-Aviv, Israel

Problem Definition

Background and Overview

Coordinating processors located in different places is one of the fundamental problems in distributed computing. In his seminal work, Lamport [4,5] studied the model where the only source of coordination is message exchange between the processors; the time that elapses between successive steps at the same processor, as well as the time spent by a message in transit, may be arbitrarily large or small. Lamport observed that in this model, called the *asynchronous model*, temporal concepts such as "past" and "future" are derivatives of causal dependence, a notion with a simple algorithmic interpretation. The work of Patt-Shamir and Rajsbaum [10] can be viewed as extending Lamport's qualitative treatment with quantitative concepts. For example, a statement like "event a happened before event b" may be refined to a statement like "event a happened at least 2 time units and at most 5 time units before event b". This is in contrast to most previous theoretical work, which focused on the linear-programming aspects of clock synchronization (see below).

The basic idea in [10] is as follows. First, the framework is extended to allow for upper and lower bounds on the time that elapses between pairs of events, using the system's *real-time specification*. The notion of real-time specification is a very natural one. For example, most processors have local clocks, whose rate of progress is typically bounded with respect to real time (these bounds are usually referred to as the clock's "drift bounds"). Another example is send and receive events of a given message: It is always true that the receive event occurs before the send event, and in many cases, tighter lower and upper bounds are available. Having defined real-time specification, [10] proceeds to show how to combine these local bounds global bounds in the best possible way using simple graph-theoretic concepts. This allows one to derive optimal protocols that say, for example, what is the current reading of a remote clock. If that remote clock is the standard clock, then the result is optimal clock synchronization in the common sense (this concept is called "external synchronization" below).

Formal Model

The system consists of a fixed set of interconnected *processors*. Each processor has a *local clock*. An *execution* of the system is a sequence of events, where each event is either a *send* event, a *receive* event, or an *internal* event. Regarding communication, it is only assumed that each receive event of a message m has a unique corresponding send event of m. This means that messages may be arbitrarily lost, duplicated or reordered, but not corrupted. Each event e occurs at a single specified processor, and has two real numbers associated with it: its *local time*, denoted $\mathrm{LT}(e)$, and its *real time*, denoted $\mathrm{RT}(e)$. The local time of an event models the reading of the local clock when that event occurs, and the local processor may use this value, e. g., for calculations, or by sending it over to another processor. By contrast, the real time of an event is not observable by processors: it is an abstract concept that exists only in the analysis.

Finally, the real-time properties of the system are modeled by a pair of functions that map each pair of events to $\mathbb{R} \cup \{-\infty, \infty\}$: given two events e and e', $L(e, e') = \ell$ means that $\mathrm{RT}(e') - \mathrm{RT}(e) \geq \ell$, and $H(e, e') = h$ means that $\mathrm{RT}(e') - \mathrm{RT}(e) \leq h$, i. e., that the number of (real) time units since the occurrence of event e until the occurrence of e' is at least ℓ and at most h. Without loss of generality, it is assumed that $L(e, e') = -H(e', e)$ for all events e, e' (just use the smaller of them). Henceforth, only the upper bounds function H is used to represent the real-time specification.

Some special cases of real time properties are particularly important. In a completely asynchronous system, $H(e', e) = 0$ if either e occurs before e' in the same processor, or if e and e' are the send and receive events, respectively, of the same message. (For simplicity, it is assumed that two ordered events may have the same real time of occurrence.) In all other cases $H(e, e') = \infty$. On the other extreme of the model spectrum, there is the *drift-free* clocks model, where all local clocks run at exactly the rate of real time. Formally, in this case $H(e, e') = \mathrm{LT}(e') - \mathrm{LT}(e)$ for any two events e and e' occurring at the same processor. Obviously, it may be the case that only some of the clocks in the system are drift-free.

Algorithms

In this work, message generation and delivery is completely decoupled from message information. Formally, messages are assumed to be generated by some "send module", and delivered by the "communication system". The task of algorithms is to add contents in messages and state variables in each node. (The idea of decoupling synchronization information from message generation was introduced in [1].) The algorithm only has local information, i. e., contents of the local state variables and the local clock, as well as the contents of the incoming message, if we are dealing with a receive event. It is also assumed that the real time specification is known to the algorithm. The conjunction of the events, their and their local times (but not their real times) is called as the *view* of the given execution. Algorithms, therefore, can only use as input the view of an execution and its real time specification.

Problem Statement

The simplest variant of clock synchronization is *external synchronization*, where one of the processors, called the source, has a drift-free clock, and the task of all processors is to maintain the tightest possible estimate on the current reading of the source clock. This formulation corresponds to the Newtonian model, where the processors reside in a well-defined time coordinate system, and the source clock is reading the standard time. Formally, in external synchronization each processor v has two output variables Δ_v and ε_v; the estimate of v of the source time at a given state is $LT_v + \Delta_v$, where LT_v is the current local time at v. The algorithm is required to guarantee that the difference between the source time and it estimate is at most ε_v (note that Δ_v, as well as ε_v, may change dynamically during the execution). The performance of the algorithm is judged by the value of the ε_v variables: the smaller, the better.

In another variant of the problem, called *internal synchronization*, there is no distinguished processor, and the requirement is essentially that all clocks will have values which are close to each other. Defining this variant is not as straightforward, because trivial solutions (e. g., "set all clocks to 0 all the time") must be disqualified.

Key Results

The key construct used in [10] is the *synchronization graph* of an execution, defined by combining the concepts of local times and real-time specification as follows.

Definition 1 Let β be a view of an execution of the system, and let H be a real time specification for β. The *synchronization graph* generated by β and H is a directed weighted graph $\Gamma_{\beta H} = (V, E, w)$, where V is the set of events in β, and for each ordered pair of events $p\,q$ in β such that $H(p, q) < \infty$, there is a directed edge $(p, q) \in E$. The *weight* of an edge (p, q) is $w(p, q) \overset{\text{def}}{=} H(p, q) - \mathrm{LT}(p) + \mathrm{LT}(q)$.

The natural concept of *distance* from an event p to an event q in a synchronization graph Γ, denoted $d_\Gamma(p, q)$, is defined by the length of the shortest weight path from p to q, or infinity if q is not reachable from p. Since weights may be negative, one has to prove that the concept is well defined: indeed, it is shown that if $\Gamma_{\beta H}$ is derived from an execution with view β that satisfies real time specification H, then $\Gamma_{\beta H}$ does not contain directed cycles of negative weight.

The main algorithmic result concerning synchronization graphs is summarized in the following theorem.

Theorem 1 *Let α be an execution with view β. Then α satisfies the real time specification H if and only if $\mathrm{RT}(p) - \mathrm{RT}(q) \le d_\Gamma(p, q) + \mathrm{LT}(p) - \mathrm{LT}(q)$ for any two events p and q in $\Gamma_{\beta H}$.*

Note that all quantities in the r.h.s. of the inequality are available to the synchronization algorithm, which can therefore determine upper bounds on the real time that elapses between events. Moreover, these bounds are the best possible, as implied by the next theorem.

Theorem 2 *Let $\Gamma_{\beta H} = (V, E, w)$ be a synchronization graph obtained from a view β satisfying real time specification H. Then for any given event $p_0 \in V$, and for any finite number $N > 0$, there exist executions α_0 and α_1 with view β, both satisfying H, and such that the following real time assignments hold.*

- *In α_0, for all $q \in V$ with $d_\Gamma(q, p_0) < \infty$, $\mathrm{RT}_{\alpha_0}(q) = \mathrm{LT}(q) + d_\Gamma(q, p_0)$, and for all $q \in V$ with $d_\Gamma(q, p_0) = \infty$, $\mathrm{RT}_{\alpha_0}(q) > \mathrm{LT}(q) + N$.*
- *In α_1, for all $q \in V$ with $d_\Gamma(p_0, q) < \infty$, $\mathrm{RT}_{\alpha_1}(q) = \mathrm{LT}(q) - d_\Gamma(p_0, q)$, and for all $q \in V$ with $d_\Gamma(p_0, q) = \infty$, $\mathrm{RT}_{\alpha_1}(q) < \mathrm{LT}(q) - N$.*

From the algorithmic viewpoint, one important drawback of results of Theorems 1 and 2 is that they depend on the view of an execution, which may grow without bound. Is it really necessary? The last general result in [10] answers this question in the affirmative. Specifically, it is shown that in some variant of the *branching program* computational model, the space complexity of any synchronization algorithm that works with arbitrary real time specifications cannot be bounded by a function of the system size. The result is proved by considering multiple scenarios on a simple system of four processors on a line.

Later Developments

Based on the concept of synchronization graph, Ostrovsky and Patt-Shamir present a refined general optimal algorithm for clock synchronization [9]. The idea in [9] is to discard parts of the synchronization graphs that are no longer relevant. Roughly speaking, the complexity of the algorithm is bounded by a polynomial in the system size and the ratio of processors speeds.

Much theoretical work was invested in the internal synchronization variant of the problem. For example, Lundelius and Lynch [7] proved that in a system of n processors with full connectivity, if message delays can take arbitrary values in $[0, 1]$ and local clocks are drift-free, then the best synchronization that can be guaranteed is $1 - \frac{1}{n}$. Helpern et al. [3] extended their result to general graphs using linear-programming techniques. This work, in turn, was extended by Attiya et al. [1] to analyze any given execution (rather than only the worst case for a given topology), but the analysis is performed off-line and in a centralized fashion. The work of Patt-Shamir and Rajsbaum [11] extended the "per execution" viewpoint to on-line distributed algorithms, and shifted the focus of the problem to external synchronization.

Recently, Fan and Lynch [2] proved that in a line of n processors whose clocks may drift, no algorithm can guarantee that the difference between the clock readings of all pairs of neighbors is $o(\log n/\log\log n)$.

Clock synchronization is very useful in practice. See, for example, Liskov [6] for some motivation. It is worth noting that the Internet provides a protocol for external clock synchronization called NTP [8].

Applications

Theorem 1 immediately gives rise to an algorithm for clock synchronization: every processor maintains a representation of the synchronization graph portion known to it. This can be done using a full information protocol: In each outgoing message this graph is sent, and whenever a message arrives, the graph is extended to include the new information from the graph in the arriving message. By Theorem 2, the synchronization graph obtained this way represents at any point in time all information available required for optimal synchronization. For example, consider external synchronization. Directly from definitions it follows that all events associated with a drift-free clock (such as events in the source node) are at distance 0 from each other in the synchronization graph, and can therefore be considered, for distance computations, as a single node s. Now, assuming that the source clock actually shows real time, it is easy to see that for any event p,

$$\mathrm{RT}(p) \in [\mathrm{LT}(p) - d(s, p), \mathrm{LT}(p) + d(p, s)] ,$$

and furthermore, no better bounds can be obtained by any correct algorithm.

The general algorithm described above (maintaining the complete synchronization graph) can be used also to obtain optimal results for internal synchronization; details are omitted.

An interesting special case is where all clocks are drift free. In this case, the size of the synchronization graph remains fixed: similarly to a source node in external synchronization, all events occurring at the same processor can be mapped to a single node; parallel edges can be replaced by a single new edge whose weight is minimal among all old edges. This way one can obtain a particularly efficient distributed algorithm solving external clock synchronization, based on the distributed Bellman–Ford algorithm for distance computation.

Finally, note that the asynchronous model may also be viewed as a special case of this general theory, where an event p "happens before" an event q if and only if $d(p, q) \leq 0$.

Open Problems

One central issue in clock synchronization is faulty executions, where the real time specification is violated. Synchronization graphs detect any detectable error: views which do not have an execution that conforms with the real time specification will result in synchronization graphs with negative cycles. However, it is desirable to overcome such faults, say by removing from the synchronization graph some edges so as to break all negative-weight cycles. The natural objective in this case is to remove the least number of edges. This problem is APX-hard as it generalizes the Feedback Arc Set problem. Unfortunately, no non-trivial approximation algorithms for it are known.

Cross References

► Causal Order, Logical Clocks, State Machine Replication

Recommended Reading

1. Attiya, H., Herzberg, A., Rajsbaum, S.: Optimal clock synchronization under different delay assumptions. SIAM J. Comput. **25**(2), 369–389 (1996)
2. Fan, R., Lynch, N.A.: Gradient clock synchronization. Distrib. Comput. **18**(4), 255–266 (2006)
3. Halpern, J.Y., Megiddo, N., Munshi, A.A.: Optimal precision in the presence of uncertainty. J. Complex. **1**, 170–196 (1985)
4. Lamport, L.: Time, clocks, and the ordering of events in a distributed system. Commun. ACM **21**(7), 558–565 (1978)
5. Lamport, L.: The mutual exclusion problem. Part I: A theory of interprocess communication. J. ACM **33**(2), 313–326 (1986)
6. Liskov, B.: Practical uses of synchronized clocks in distributed systems. Distrib. Comput. **6**, 211–219 (1993). Invited talk at the 9th Annual ACM Symposium on Principles of Distributed Computing, Quebec City 22–24 August 1990
7. Lundelius, J., Lynch, N.: A new fault-tolerant algorithm for clock synchronization. Inf. Comput. **77**, 1–36 (1988)
8. Mills, D.L.: Computer Network Time Synchronization: The Network Time Protocol. CRC Press, Boca Raton (2006)
9. Ostrovsky, R., Patt-Shamir, B.: Optimal and efficient clock synchronization under drifting clocks. In: Proceedings of the 18th Annual Symposium on Principles of Distributed Computing, pp. 3–12, Atlanta, May (1999)
10. Patt-Shamir, B., Rajsbaum, S.: A theory of clock synchronization. In: Proceedings of the 26th Annual ACM Symposium on Theory of Computing, pp. 810–819, Montreal, May (1994)
11. Patt-Shamir, B., Rajsbaum, S.: A theory of clock synchronization. In: Proceedings of the 26th Annual ACM Symposium on Theory of Computing, pp. 810–819, Montreal 23–25 May 1994

Closest String and Substring Problems

2002; Li, Ma, Wang

LUSHENG WANG
Department of Computer Science,
City University of Hong Kong, Hong Kong, China

Problem Definition

The problem of finding a center string that is "close" to every given string arises and has applications in computational molecular biology and coding theory.

This problem has two versions: The first problem comes from coding theory when we are looking for a code not too far away from a given set of codes.

Problem 1 (The closest string problem)
INPUT: *a set of strings $S = \{s_1, s_2, \dots, s_n\}$, each of length m.*
OUTPUT: *the smallest d and a string s of length m which is within Hamming distance d to each $s_i \in S$.*

The second problem is much more elusive than the Closest String problem. The problem is formulated from applications in finding conserved regions, genetic drug target identification, and genetic probes in molecular biology.

Problem 2 (The closest substring problem)
INPUT: *an integer L and a set of strings $S = \{s_1, s_2, \dots, s_n\}$, each of length m.*
OUTPUT: *the smallest d and a string s, of length L, which is within Hamming distance d away from a length L substring t_i of s_i for $i = 1, 2, \dots n$.*

Key Results

The following results are from [1].

Theorem 1 *There is a polynomial time approximation scheme for the closest string problem.*

Theorem 2 *There is a polynomial time approximation scheme for the closest substring problem.*

Results for other measures can be found in [10,11,12].

Applications

Many problems in molecular biology involve finding similar regions common to each sequence in a given set of DNA, RNA, or protein sequences. These problems find applications in locating binding sites and finding conserved regions in unaligned sequences [2,7,9,13,14], genetic drug target identification [8], designing genetic probes [8], universal PCR primer design [4,8], and, outside computational biology, in coding theory [5,6]. Such problems may be considered to be various generalizations of the common substring problem, allowing errors. Many measures have been proposed for finding such regions common to every given string. A popular and one of the most fundamental measures is the Hamming distance. Moreover, two popular objective functions are used in these areas. One is the total sum of distances between the center string (common substring) and each of the given strings. The other is the maximum distance between the center string and a given string. For more details, see [8].

A more General Problem

The *distinguishing substring selection* problem has as input two sets of strings, B and G. It is required to find a substring of unspecified length (denoted by L) such that it is, informally, close to a substring of every string in B and far away from every length L substring of strings in $\mathcal{G}$. However, we can go through all the possible L and we may assume that every string in $\mathcal{G}$ has the same length L since $\mathcal{G}$ can be reconstructed to contain all substrings of length L in each of the good strings.

The problem is formally defined as follows: Given a set $\mathcal{B} = \{s_1, s_2, \ldots, s_{n_1}\}$ of n_1 (bad) strings of length at least L, and a set $\mathcal{G} = \{g_1, g_2, \ldots g_{n_2}\}$ of n_2 (good) strings of length exactly L, as well as two integers d_b and d_g ($d_b \le d_g$), the distinguishing substring selection problem (*DSSP*) is to find a string s such that for each string $s_i \in \mathcal{B}$ there exists a length-L substring t_i of s_i with $d(s, t_i) \le d_b$ and for any string $g_i \in \mathcal{G}$, $d(s, g_i) \ge d_g$. Here $d(,)$ represents the Hamming distance between two strings. If all strings in $\mathcal{B}$ are also of the same length L, the problem is called the distinguishing string problem (DSP).

The distinguishing string problem was first proposed in [8] for generic drug target design. The following results are from [3].

Theorem 3 *There is a polynomial time approximation scheme for the distinguishing substring selection problem. That is, for any constant $\epsilon > 0$, the algorithm finds a string s of length L such that for every $s_i \in \mathcal{B}$, there is a length-L substring t_i of s_i with $d(t_i, s) \le (1+\epsilon)d_b$ and for every substring u_i of length L of every $g_i \in \mathcal{G}$, $d(u_i, s) \ge (1-\epsilon)d_g$, if a solution to the original pair ($d_b \le d_g$) exists. Since there are a polynomial number of such pairs (d_b, d_g), we can exhaust all the possibilities in polynomial time to find a good approximation required by the corresponding application problems.*

Open Problems

The PTAS's designed here use linear programming and randomized rounding technique to solve some cases for the problem. Thus, the running time complexity of the algorithms for both the closest string and closest substring is very high. An interesting open problem is to design more efficient PTAS's for both problems.

Cross References

► Closest Substring
► Efficient Methods for Multiple Sequence Alignment with Guaranteed Error Bounds
► Engineering Algorithms for Computational Biology
► Multiplex PCR for Gap Closing (Whole-genome Assembly)

Recommended Reading

1. Ben-Dor, A., Lancia, G., Perone, J., Ravi, R.: Banishing bias from consensus sequences. In: Proc. 8th Ann. Combinatorial Pattern Matching Conf., pp. 247–261. (1997)
2. Deng, X., Li, G., Li, Z., Ma, B., Wang, L.: Genetic Design of Drugs Without Side-Effects. SIAM. J. Comput. **32**(4), 1073–1090 (2003)
3. Dopazo, J., Rodríguez, A., Sáiz, J.C., Sobrino, F.: Design of primers for PCR amplification of highly variable genomes. CABIOS **9**, 123–125 (1993)
4. Frances, M., Litman, A.: On covering problems of codes. Theor. Comput. Syst. **30**, 113–119 (1997)
5. Gąsieniec, L., Jansson, J., Lingas, A.: Efficient approximation algorithms for the hamming center problem. In: Proc. 10th ACM-SIAM Symp. on Discrete Algorithms., pp. 135–S906. (1999)
6. Hertz, G., Stormo, G.: Identification of consensus patterns in unaligned DNA and protein sequences: a large-deviation statistical basis for penalizing gaps. In: Proc. 3rd Int'l Conf. Bioinformatics and Genome Research, pp. 201–216. (1995)
7. Lanctot, K., Li, M., Ma, B., Wang, S., Zhang, L.: Distinguishing string selection problems. In: Proc. 10th ACM-SIAM Symp. on Discrete Algorithms, pp. 633–642. (1999)
8. Lawrence, C., Reilly, A.: An expectation maximization (EM) algorithm for the identification and characterization of common sites in unaligned biopolymer sequences. Proteins **7**, 41–51 (1990)
9. Li, M., Ma, B., Wang, L.: On the closest string and substring problems. J. ACM **49**(2), 157–171 (2002)
10. Li, M., Ma, B., Wang, L.: Finding similar regions in many sequences. J. Comput. Syst. Sci. (1999)
11. Li, M., Ma, B., Wang, L.: Finding similar regions in many strings. In: Proceedings of the Thirty-first Annual ACM Symposium on Theory of Computing, pp. 473–482. Atlanta (1999)
12. Ma, B.: A polynomial time approximation scheme for the closest substring problem. In: Proc. 11th Annual Symposium on Combinatorial Pattern Matching, Montreal, pp. 99–107. (2000)
13. Stormo, G.: Consensus patterns in DNA. In: Doolittle, R.F. (ed.) Molecular evolution: computer analysis of protein and nucleic acid sequences. Methods in Enzymology, vol. 183, pp. 211–221 (1990)
14. Stormo, G., Hartzell III, G.W.: Identifying protein-binding sites from unaligned DNA fragments. Proc. Natl. Acad. Sci. USA. **88**, 5699–5703 (1991)

Closest Substring

2005; Marx

Jens Gramm
WSI Institute of Theoretical Computer Science,
Tübingen University, Tübingen, Germany

Keywords and Synonyms

Common approximate substring

Problem Definition

CLOSEST SUBSTRING is a core problem in the field of consensus string analysis with, in particular, applications in computational biology. Its decision version is defined as follows.

CLOSEST SUBSTRING
Input: k strings $s_1, s_2, \dots, s_k$ over alphabet Σ and non-negative integers d and L.
Question: Is there a string s of length L and, for all $i = 1, \dots, k$, a length-L substring s_i' of s_i such that $d_H(s, s_i') \le d$?

Here $d_H(s, s_i')$ denotes the Hamming distance between s and $s_i{}'$, i. e., the number of positions in which s and $s_i{}'$ differ. Following the notation used in [7], m is used to denote the average length of the input strings and n to denote the total size of the problem input.

The optimization version of CLOSEST SUBSTRING asks for the minimum value of the distance parameter d for which the input strings still allow a solution.

Key Results

The classical complexity of CLOSEST SUBSTRING is given by

Theorem 1 ([4,5]) CLOSEST SUBSTRING *is NP-complete, and remains so for the special case of the* CLOSEST STRING *problem, where the requested solution string s has to be of same length as the input strings.* CLOSEST STRING *is NP-complete even for the further restriction to a binary alphabet.*

The following theorem gives the central statement concerning the problem's approximability:

Theorem 2 ([6]) CLOSEST SUBSTRING *(as well as* CLOSEST STRING*) admit polynomial time approximation schemes (PTAS's), where the objective function is the minimum Hamming distance d.*

In its randomized version, the PTAS cited by Theorem 2 computes, with high probability, a solution with Hamming distance $(1+\epsilon)d_{\text{opt}}$ for an optimum value d_{opt} in $(k^2 m)^{O(\log|\Sigma|/\epsilon^4)}$ running time. With additional overhead, this randomized PTAS can be derandomized. A straightforward and efficient factor-2 approximation for CLOSEST STRING is obtained by trying all length-L substrings of one of the input strings.

The following two statements address the problem's parametrized complexity, with respect to both obvious problem parameters d and k:

Theorem 3 ([3]) CLOSEST SUBSTRING *is W[1]-hard with respect to the parameter k, even for binary alphabet.*

Theorem 4 ([7]) CLOSEST SUBSTRING *is W[1]-hard with respect to the parameter d, even for binary alphabet.*

For non-binary alphabet the statement of Theorem 3 has been shown independently by Evans et al. [2]. Theorems 3 and 4 show that an exact algorithm for CLOSEST SUBSTRING with polynomial running time is unlikely for a constant value of d as well as for a constant value of k, i. e. such an algorithm does not exist unless 3-SAT can be solved in subexponential time.

Theorem 4 also allows additional insights into the problem's approximability: In the PTAS for CLOSEST SUBSTRING, the exponent of the polynomial bounding the running time depends on the approximation factor. These are not "efficient" PTAS's (EPTAS's), i. e. PTAS's with a $f(\epsilon) \cdot n^c$ running time for some function f and some constant c, and therefore are probably not useful in practice. Theorem 4 implies that most likely the PTAS with the $n^{O(1/\epsilon^4)}$ running time presented in [6] cannot be improved to an EPTAS. More precisely, there is no $f(\epsilon) \cdot n^{o(\log 1/\epsilon)}$ time PTAS for CLOSEST SUBSTRING unless 3-SAT can be solved in subexponential time. Moreover, the proof of Theorem 4 also yields

Theorem 5 ([7]) *There are no $f(d,k) \cdot n^{o(\log d)}$ time and no $g(d,k) \cdot n^{o(\log\log k)}$ exact algorithms solving* CLOSEST SUBSTRING *for some functions f and g unless* 3-SAT *can be solved in subexponential time.*

For unbounded alphabet the bounds have been strengthened by showing that Closest Substring has no PTAS with running time $f(\epsilon) \cdot n^{o(1/\epsilon)}$ for any function f unless 3-SAT can be solved in subexponential time [10]. The following statements provide exact algorithms for CLOSEST SUBSTRING with small fixed values of d and k, matching the bounds given in Theorem 5:

Theorem 6 ([7]) CLOSEST SUBSTRING *can be solved in time $f(d) \cdot n^{O(\log d)}$ for some function f, where, more precisely, $f(d) = |\Sigma|^{d(\log d+2)}$.*

Theorem 7 ([7]) CLOSEST SUBSTRING *can be solved in time $g(d,k) \cdot n^{O(\log\log k)}$ for some function g, where, more precisely, $g(d,k) = (|\Sigma|d)^{O(kd)}$.*

With regard to problem parameter L, CLOSEST SUBSTRING can be trivially solved in $O(|\Sigma|^L \cdot n)$ time by trying all possible strings over alphabet Σ.

Applications

An application of CLOSEST SUBSTRING lies in the analysis of biological sequences. In motif discovery, a goal is to search "signals" common to a set of selected strings representing DNA or protein sequences. One way to represent these signals are approximately preserved substrings occurring in each of the input strings. Employing Hamming distance as a biologically meaningful distance measure results in the problem formulation of CLOSEST SUBSTRING.

For example, Sagot [9] studies motif discovery by solving CLOSEST SUBSTRING (and generalizations thereof) using suffix trees; this approach has a worst-case running time of $O(k^2 m \cdot L^d \cdot |\Sigma|^d)$. In the context of motif discovery, also heuristics applicable to CLOSEST SUBSTRING were proposed, e. g., Pevzner and Sze [8] present an algorithm called WINNOWER and Buhler and Tompa [1] use a technique called random projections.

Open Problems

It is open [7] whether the $n^{O(1/\epsilon^4)}$ running time of the approximation scheme presented in [6] can be improved to $n^{O(\log 1/\epsilon)}$, matching the bound derived from Theorem 4.

Cross References

The following problems are close relatives of CLOSEST SUBSTRING:

- ► Closest String is the special case of CLOSEST SUBSTRING, where the requested solution string s has to be of same length as the input strings.
- Distinguishing Substring Selection is the generalization of CLOSEST SUBSTRING, where a second set of input strings and an additional integer d' are given and where the requested solution string s has – in addition to the requirements posed by CLOSEST SUBSTRING – Hamming distance at least d' with every length-L substring from the second set of strings.
- Consensus Patterns is the problem obtained by replacing, in the definition of CLOSEST SUBSTRING, the maximum of Hamming distances by the sum of Hamming distances. The resulting modified question of CONSENSUS PATTERNS is: Is there a string s of length L with

$$\sum_{i=1,\ldots,m} d_H(s, s'_i) \leq d?$$

CONSENSUS PATTERNS is the special case of SUBSTRING PARSIMONY in which the phylogenetic tree provided in the definition of SUBSTRING PARSIMONY is a star phylogeny.

Recommended Reading

1. Buhler, J., Tompa, M.: Finding motifs using random projections. J. Comput. Biol. **9**(2), 225–242 (2002)
2. Evans, P.A., Smith, A.D., Wareham, H.T.: On the complexity of finding common approximate substrings. Theor. Comput. Sci. **306**(1–3), 407–430 (2003)
3. Fellows, M.R., Gramm, J., Niedermeier, R.: On the parameterized intractability of motif search problems. Combinatorica **26**(2), 141–167 (2006)
4. Frances, M., Litman, A.: On covering problems of codes. Theor. Comput. Syst. **30**, 113–119 (1997)
5. Lanctot, J.K.: Li, M., Ma, B., Wang, S., Zhang, L.: Distinguishing String Search Problems. Inf. Comput. **185**, 41–55 (2003)
6. Li, M., Ma, B., Wang, L.: On the Closest String and Substring Problems. J. ACM **49**(2), 157–171 (2002)
7. Marx, D.: The Closest Substring problem with small distances. In: Proceedings of the 46th FOCS, pp 63–72. IEEE Press, (2005)
8. Pevzner, P.A., Sze, S.H.: Combinatorial approaches to finding subtle signals in DNA sequences. In: Proc. of 8th ISMB, pp. 269–278. AAAI Press, (2000)
9. Sagot, M.F.: Spelling approximate repeated or common motifs using a suffix tree. In: Proc. of the 3rd LATIN, vol. 1380 in LNCS, pp. 111–127. Springer (1998)
10. Wang, J., Huang, M., Cheng, J.: A Lower Bound on Approximation Algorithms for the Closest Substring Problem. In: Proceedings COCOA 2007, vol. 4616 in LNCS, pp. 291–300 (2007)

Clustering

► Local Search for K-medians and Facility Location
► Well Separated Pair Decomposition for Unit–Disk Graph

Color Coding

1995; Alon, Yuster, Zwick

NOGA ALON[1], RAPHAEL YUSTER[2], URI ZWICK[3]
[1] Department of Mathematics and Computer Science, Tel-Aviv University, Tel-Aviv, Israel
[2] Department of Mathematics, University of Haifa, Haifa, Israel
[3] Department of Mathematics and Computer Science, Tel-Aviv University, Tel-Aviv, Israel

Keywords and Synonyms

Finding small subgraphs within large graphs

Problem Definition

Color coding [2] is a novel method used for solving, in polynomial time, various subcases of the generally NP-Hard *subgraph isomorphism* problem. The input for the subgraph isomorphism problem is an ordered pair of (possibly directed) graphs (G, H). The output is either a mapping showing that H is isomorphic to a (possibly induced) subgraph of G, or **false** if no such subgraph exists. The subgraph isomorphism problem includes, as special cases, the HAMILTON-PATH, CLIQUE, and INDEPENDENT SET problems, as well as many others. The problem is also interesting when H is *fixed*. The goal, in this case, is to design algorithms whose running times are significantly better than the running time of the naïve algorithm.

Method Description

The color coding method is a randomized method. The vertices of the graph $G = (V, E)$ in which a subgraph isomorphic to $H = (V_H, E_H)$ is sought are randomly colored by $k = |V_H|$ colors. If $|V_H| = O(\log |V|)$, then with a small probability, but only polynomially small (i. e., one over a polynomial), all the vertices of a subgraph of G which is isomorphic to H, if there is such a subgraph, will be colored by distinct colors. Such a subgraph is called *color coded*. The color coding method exploits the fact that, in many cases, it is easier to detect color coded subgraphs than uncolored ones.

Perhaps the simplest interesting subcases of the subgraph isomorphism problem are the following: Given a directed or undirected graph $G = (V, E)$ and a number k, does G contain a simple (directed) path of length k? Does G contain a simple (directed) cycle of length *exactly* k? The following describes a $2^{O(k)} \cdot |E|$ time algorithm that receives as input the graph $G = (V, E)$, a coloring $c\colon V \to \{1, \dots, k\}$ and a vertex $s \in V$, and finds a colorful path of length $k - 1$ that starts at s, if one exists. To find a colorful path of length $k - 1$ in G that starts somewhere, just add a new vertex s' to V, color it with a new color 0 and connect it with edges to all the vertices of V. Now look for a colorful path of length k that starts at s'.

A colorful path of length $k - 1$ that starts at some specified vertex s is found using a dynamic programming approach. Suppose one is already given, for each vertex $v \in V$, the possible sets of colors on colorful paths of length i that connect s and v. Note that there is no need to record all colorful paths connecting s and v. Instead, record the color sets appearing on such paths. For each vertex v there is a collection of at most $\binom{k}{i}$ color sets. Now, inspect every subset C that belongs to the collection of v, and every edge $(v, u) \in E$. If $c(u) \notin C$, add the set $C \cup \{c(u)\}$ to the collection of u that corresponds to colorful paths of length $i + 1$. The graph G contains a colorful path of length $k - 1$ with respect to the coloring c if and only if the final collection, that corresponding to paths of length $k - 1$, of at least one vertex is non-empty. The number of operations performed by the algorithm outlined is at most $O(\sum_{i=0}^{k} i\binom{k}{i} \cdot |E|)$ which is clearly $O(k2^k \cdot |E|)$.

Derandomization

The randomized algorithms obtained using the color coding method are derandomized with only a small loss in efficiency. All that is needed to derandomize them is a family of colorings of $G = (V, E)$ so that every subset of k vertices of G is assigned distinct colors by at least one of these colorings. Such a family is also called a family of *perfect hash functions* from $\{1, 2, \dots, |V|\}$ to $\{1, 2, \dots, k\}$. Such a family is explicitly constructed by combining the methods of [1,9,12,16]. For a derandomization technique yielding a constant factor improvement see [5].

Key Results

Lemma 1 *Let $G = (V, E)$ be a directed or undirected graph and let $c\colon V \to \{1, \dots, k\}$ be a coloring of its vertices with k colors. A colorful path of length $k - 1$ in G, if one exists, can be found in $2^{O(k)} \cdot |E|$ worst-case time.*

Lemma 2 *Let $G = (V, E)$ be a directed or undirected graph and let $c\colon V \to \{1, \dots, k\}$ be a coloring of its vertices with k colors. All pairs of vertices connected by colorful paths of length $k - 1$ in G can be found in either $2^{O(k)} \cdot |V||E|$ or $2^{O(k)} \cdot |V|^{\omega}$ worst-case time (here $\omega < 2.376$ denotes the matrix multiplication exponent).*

Using the above lemmata the following results are obtained.

Theorem 3 *A simple directed or undirected path of length $k - 1$ in a (directed or undirected) graph $G = (V, E)$ that contains such a path can be found in $2^{O(k)} \cdot |V|$ expected time in the undirected case and in $2^{O(k)} \cdot |E|$ expected time in the directed case.*

Theorem 4 *A simple directed or undirected cycle of size k in a (directed or undirected) graph $G = (V, E)$ that contains such a cycle can be found in either $2^{O(k)} \cdot |V||E|$ or $2^{O(k)} \cdot |V|^{\omega}$ expected time.*

A cycle of length k in minor-closed families of graphs can be found, using color coding, even faster (for planar graphs, a slightly faster algorithm appears in [6]).

Theorem 5 *Let C be a non-trivial minor-closed family of graphs and let $k \geq 3$ be a fixed integer. Then, there exists a randomized algorithm that given a graph $G = (V, E)$ from C, finds a C_k (a simple cycle of size k) in G, if one exists, in $O(|V|)$ expected time.*

As mentioned above, all these theorems can be derandomized at the price of a $\log|V|$ factor. The algorithms are also easily to parallelize.

Applications

The initial goal was to obtain efficient algorithms for finding simple paths and cycles in graphs. The color coding method turned out, however, to have a much wider range of applicability. The linear time (i. e., $2^{O(k)} \cdot |E|$ for directed graphs and $2^{O(k)} \cdot |V|$ for undirected graphs) bounds for simple paths apply in fact to any *forest* on k vertices. The $2^{O(k)} \cdot |V|^{\omega}$ bound for simple cycles applies in fact to any *series-parallel* graph on k vertices. More generally, if $G = (V, E)$ contains a subgraph isomorphic to a graph $H = (V_H, E_H)$ whose *tree-width* is at most t, then such a subgraph can be found in $2^{O(k)} \cdot |V|^{t+1}$ expected time, where $k = |V_H|$. This improves an algorithm of Plehn and Voigt [14] that has a running time of $k^{O(k)} \cdot |V|^{t+1}$. As a very special case, it follows that the LOG PATH problem is in P. This resolves in the affirmative a conjecture of Papadimitriou and Yannakakis [13]. The exponential dependence on k in the above bounds is probably unavoidable as the problem is NP-complete if k is part of the input.

The color coding method has been a fruitful method in the study of parametrized algorithms and parametrized complexity [7,8]. Recently, the method has found interesting applications in computational biology, specifically in detecting signaling pathways within protein interaction networks, see [10,17,18,19].

Open Problems

Several problems, listed below, remain open.

- Is there a polynomial time (deterministic or randomized) algorithm for deciding if a given graph $G = (V, E)$ contains a path of length, say, $\log^2|V|$? (This is unlikely, as it will imply the existence of an algorithm that decides in time $2^{O(\sqrt{n})}$ whether a given graph on n vertices is Hamiltonian.)
- Can the $\log|V|$ factor appearing in the derandomization be omitted?
- Is the problem of deciding whether a given graph $G = (V, E)$ contains a triangle as difficult as the Boolean multiplication of two $|V| \times |V|$ matrices?

Experimental Results

Results of running the basic algorithm on biological data have been reported in [17,19].

Cross References

- ▶ Approximation Schemes for Planar Graph Problems
- ▶ Graph Isomorphism
- ▶ Treewidth of Graphs

Recommended Reading

1. Alon, N., Goldreich, O., Håstad, J., Peralta, R.: Simple constructions of almost k-wise independent random variables. Random Struct. Algorithms **3**(3), 289–304 (1992)
2. Alon, N., Yuster, R., Zwick, U.: Color coding. J. ACM **42**, 844–856 (1995)
3. Alon, N., Yuster, R., Zwick, U.: Finding and counting given length cycles. Algorithmica **17**(3), 209–223 (1997)
4. Björklund, A., Husfeldt, T.: Finding a path of superlogarithmic length. SIAM J. Comput. **32**(6), 1395–1402 (2003)
5. Chen, J., Lu, S., Sze, S., Zhang, F.: Improved algorithms for path, matching, and packing problems. Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 298–307 (2007)
6. Eppstein, D.: Subgraph isomorphism in planar graphs and related problems. J. Graph Algorithms Appl. **3**(3), 1–27 (1999)
7. Fellows, M.R.: New Directions and new challenges in algorithm design and complexity, parameterized. In: Lecture Notes in Computer Science, vol. 2748, p. 505–519 (2003)
8. Flum, J., Grohe, M.: The Parameterized complexity of counting problems. SIAM J. Comput. **33**(4), 892–922 (2004)
9. Fredman, M.L., J.Komlós, Szemerédi, E.: Storing a sparse table with $O(1)$ worst case access time. J. ACM **31**, 538–544 (1984)
10. Hüffner, F., Wernicke, S., Zichner, T.: Algorithm engineering for Color Coding to facilitate Signaling Pathway Detection. In: Proceedings of the 5th Asia-Pacific Bioinformatics Conference (APBC), pp. 277–286 (2007)
11. Monien, B.: How to find long paths efficiently. Ann. Discret. Math. **25**, 239–254 (1985)
12. Naor, J., Naor, M.: Small-bias probability spaces: efficient constructions and applications. SIAM J. Comput. Comput. **22**(4), 838–856 (1993)
13. Papadimitriou, C.H., Yannakakis, M.: On limited nondeterminism and the complexity of the V-C dimension. J. Comput. Syst. Sci. **53**(2), 161–170 (1996)
14. Plehn, J., Voigt, B.: Finding minimally weighted subgraphs. Lect. Notes Comput. Sci. **484**, 18–29 (1990)
15. Robertson, N., Seymour, P.: Graph minors. II. Algorithmic aspects of tree-width. J. Algorithms **7**, 309–322 (1986)
16. Schmidt, J.P., Siegel, A.: The spatial complexity of oblivious k-probe hash functions. SIAM J. Comput. **19**(5), 775–786 (1990)
17. Scott, J., Ideker, T., Karp, R.M., Sharan, R.: Efficient Algorithms for Detecting Signaling Pathways in Protein Interaction Networks. J. Comput. Biol. **13**(2), 133–144 (2006)
18. Sharan, R., Ideker, T.: Modeling cellular machinery through biological network comparison. Nat. Biotechnol. **24**, 427–433 (2006)
19. Shlomi, T., Segal, D., Ruppin, E., Sharan, R.: QPath: a method for querying pathways in a protein-protein interaction network. BMC Bioinform. **7**, 199 (2006)

Communication in Ad Hoc Mobile Networks Using Random Walks

2003; Chatzigiannakis, Nikoletseas, Spirakis

IOANNIS CHATZIGIANNAKIS
Department of Computer Engineering and Informatics, University of Patras and Computer Technology Institute, Patras, Greece

Keywords and Synonyms

Disconnected ad hoc networks; Delay-tolerant networks; Message Ferrying; Message relays; Data mules; Sink mobility

Problem Definition

A mobile ad hoc network is a temporary dynamic interconnection network of wireless mobile nodes without any established infrastructure or centralized administration. A *basic communication problem*, in ad hoc mobile networks, is to send information from a *sender* node, A, to another designated *receiver* node, B. If mobile nodes A and B come within wireless range of each other, then they are able to communicate. However, if they do not, they can communicate if other network nodes of the network are willing to forward their packets. One way to solve this problem is the protocol of notifying every node that the sender A meets and provide it with *all the information* hoping that some of them will eventually meet the receiver B.

> Is there a more efficient technique (other than notifying every node that the sender meets, in the hope that some of them will then eventually meet the receiver) that will effectively solve the communication establishment problem without flooding the network and exhausting the battery and computational power of the nodes?

The problem of communication among mobile nodes is one of the most fundamental problems in ad hoc mobile networks and is at the core of many algorithms, such as for counting the number of nodes, electing a leader, data processing etc. For an exposition of several important problems in ad hoc mobile networks see [13]. The work of Chatzigiannakis, Nikoletseas and Spirakis [5] focuses on wireless mobile networks that are subject to highly dynamic structural changes created by mobility, channel fluctuations and device failures. These changes affect topological connectivity, occur with high frequency and may not be predictable in advance. Therefore, the environment where the nodes move (in three-dimensional space with possible obstacles) as well as the motion that the nodes perform are *input* to any distributed algorithm.

The Motion Space

The space of possible motions of the mobile nodes is combinatorially abstracted by a *motion-graph*, i. e. the detailed geometric characteristics of the motion are neglected. Each mobile node is assumed to have a transmission range represented by a sphere tr centered by itself. Any other node inside tr can receive any message broadcast by this node. This sphere is approximated by a cube tc with volume $\mathcal{V}(tc)$, where $\mathcal{V}(tc) < \mathcal{V}(tr)$. The size of tc can be chosen in such a way that its volume $\mathcal{V}(tc)$ is the maximum that preserves $\mathcal{V}(tc) < \mathcal{V}(tr)$, and if a mobile node inside tc broadcasts a message, this message is received by any other node in tc. Given that the mobile nodes are moving in the space S, S is divided into consecutive cubes of volume $\mathcal{V}(tc)$.

Definition 1 The motion graph $G(V, E)$, $(|V| = n, |E| = m)$, which corresponds to a quantization of S is constructed in the following way: a vertex $u \in G$ represents a cube of volume $\mathcal{V}(tc)$ and an edge $(u, v) \in G$ exists if the corresponding cubes are adjacent.

The number of vertices n, actually approximates the ratio between the volume $\mathcal{V}(S)$ of space S, and the space occupied by the transmission range of a mobile node $\mathcal{V}(tr)$. In the extreme case where $\mathcal{V}(S) \approx \mathcal{V}(tr)$, the transmission range of the nodes approximates the space where they are moving and $n = 1$. Given the transmission range tr, n depends linearly on the volume of space S regardless of the choice of tc, and $n = O(V(S)/V(tr))$. The ratio $V(S)/V(tr)$ is the *relative motion space size* and is denoted by ρ. Since the edges of G represent neighboring polyhedra each vertex is connected with a constant number of neighbors, which yields that $m = \Theta(n)$. In this example where tc is a cube, G has maximum degree of six and $m \leq 6n$. Thus *motion graph* G is (usually) a *bounded degree graph* as it is derived from a regular graph of small degree by deleting parts of it corresponding to motion or communication obstacles. Let Δ be the maximum vertex degree of G.

The Motion of the Nodes-Adversaries

In the general case, the motions of the nodes are decided by an *oblivious adversary*: The adversary determines motion patterns in any possible way but independently of the distributed algorithm. In other words, the case where some of the nodes are deliberately trying to *maliciously affect* the protocol, e. g. avoid certain nodes, are excluded. This is a pragmatic assumption usually followed by applications. Such kind of motion adversaries are called *restricted motion adversaries*.

For purposes of studying efficiency of distributed algorithms for ad hoc networks *on the average*, the motions of the nodes are modeled by *concurrent and independent random walks*. The assumption that the mobile nodes move randomly, either according to uniformly distributed changes in their directions and velocities or according to the random waypoint mobility model by picking random destinations, has been used extensively by other research.

Key Results

The key idea is to take advantage of the mobile nodes natural movement by exchanging information whenever mobile nodes meet incidentally. It is evident, however, that if the nodes are spread in remote areas and they do not move beyond these areas, there is no way for information to reach them, unless the protocol takes special care of such situations. The work of Chatzigiannakis, Nikoletseas and Spirakis [5] proposes the idea of forcing only a small subset of the deployed nodes to move as per the needs of the protocol; they call this subset of nodes the *support* of the network. Assuming the availability of such nodes, they are used to provide a simple, correct and efficient strategy for communication between any pair of nodes of the network that avoids message flooding.

Let k nodes be a predefined set of nodes that become the nodes of the support. These nodes move randomly and fast enough so that they visit in sufficiently short time the entire motion graph. When some node of the support is within transmission range of a sender, it notifies the sender that it may send its message(s). The messages are then stored "somewhere within the support structure". When a receiver comes within transmission range of a node of the support, the receiver is notified that a message is "waiting" for him and the message is then forwarded to the receiver.

Protocol 1 (The "Snake" Support Motion Coordination Protocol) Let $S_0, S_1, \ldots, S_{k-1}$ be the members of the support and let S_0 denote the leader node (possibly elected). The protocol forces S_0 to perform a random walk on the motion graph and each of the other nodes S_i execute the simple protocol "move where S_{i-1} was before". When S_0 is about to move, it sends a message to S_1 that states the new direction of movement. S_1 will change its direction as per instructions of S_0 and will propagate the message to S_2. In analogy, S_i will follow the orders of S_{i-1} after transmitting the new directions to S_{i+1}. Movement orders received by S_i are positioned in a queue Q_i for sequential processing. The very first move of S_i, $\forall i \in \{1, 2, \ldots, k-1\}$ is delayed by a δ period of time.

The purpose of the random walk of the head S_0 is to ensure a *cover*, within some finite time, of the whole graph G without knowledge and memory, other than local, of topology details. This memoryless motion also ensures fairness, low-overhead and inherent robustness to structural changes.

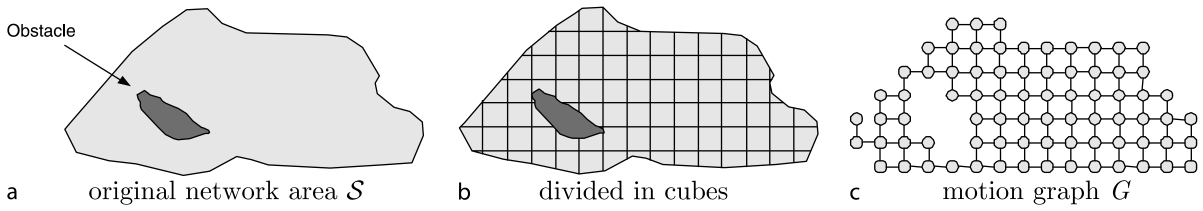
Consider the case where any sender or receiver is allowed a general, unknown motion strategy, but its strategy is provided by a restricted motion adversary. This means that each node not in the support either (a) executes a deterministic motion which either stops at a vertex or cycles forever after some initial part or (b) it executes a stochastic strategy which however is *independent* of the motion of the support. The authors in [5] prove the following correctness and efficiency results. The reader can refer to the excellent book by Aldous and Fill [1] for a nice introduction on Makrov Chains and Random Walks.

Theorem 1 *The support and the "snake" motion coordination protocol guarantee reliable communication between any sender-receiver (A, B) pair in finite time, whose expected value is bounded only by a function of the relative motion space size ρ and does not depend on the number of nodes, and is also independent of how MH_S, MH_R move, provided that the mobile nodes not in the support do not deliberately try to avoid the support.*

Theorem 2 *The expected communication time of the support and the "snake" motion coordination protocol is bounded above by $\Theta(\sqrt{mc})$ when the (optimal) support size $k = \sqrt{2mc}$ and c is $e/(e-1)u$, u being the "separation threshold time" of the random walk on G.*

Theorem 3 *By having the support's head move on a regular spanning subgraph of G, there is an absolute constant $\gamma > 0$ such that the expected meeting time of A (or B) and the support is bounded above by $\gamma n^2/k$. Thus the protocol guarantees a total expected communication time of $\Theta(\rho)$, independent of the total number of mobile nodes, and their movement.*

The analysis assumes that the head S_0 moves according to a continuous time random walk of total rate 1 (rate of exit out of a node of G). If S_0 moves ψ *times faster* than the rest of the nodes, all the estimated times, except the inter-support time, will be divided by ψ. Thus the expected total communication time can be made to be as small as $\Theta(\gamma\rho/\sqrt{\psi})$ where γ is an absolute constant. In cases where S_0 can take advantage of the network topology, all the estimated times, except the inter-support time are improved:



Communication in Ad Hoc Mobile Networks Using Random Walks, Figure 1
The original network area $\mathcal{S}$ (a), how it is divided in consecutive cubes of volume $\mathcal{V}(tc)$ (b) and the resulting motion graph G (c)

Theorem 4 *When the support's head moves on a regular spanning subgraph of G the expected meeting time of A (or B) and the support cannot be less than $(n-1)^2/2m$. Since $m = \Theta(n)$, the lower bound for the expected communication time is $\Theta(n)$. In this sense, the "snake" protocol's expected communication time is optimal, for a support size which is $\Theta(n)$.*

The "on-the-average" analysis of the time-efficiency of the protocol assumes that the motion of the mobile nodes not in the support *is a random walk on the motion graph G*. The random walk of each mobile node is performed independently of the other nodes.

Theorem 5 *The expected communication time of the support and the "snake" motion coordination protocol is bounded above by the formula*

$$E(T) \leq \frac{2}{\lambda_2(G)}\Theta\left(\frac{n}{k}\right) + \Theta(k)\,.$$

The upper bound is minimized when $k = \sqrt{2n/\lambda_2(G)}$, where λ_2 is the second eigenvalue of the motion graph's adjacency matrix.

The way the support nodes move and communicate is robust, in the sense that it can tolerate failures of the support nodes. The types of failures of nodes considered are permanent, i. e. stop failures. Once such a fault happens, the support node of the fault does not participate in the ad hoc mobile network anymore. A communication protocol is *β-faults tolerant*, if it still allows the members of the network to communicate correctly, under the presence of at most β permanent faults of the nodes in the support ($\beta \geq 1$). [5] shows that:

Theorem 6 *The support and the "snake" motion coordination protocol is 1-fault tolerant.*

Applications

Ad hoc mobile networks are rapidly deployable and self-configuring networks that have important applications in many critical areas such as disaster relief, ambient intelligence, wide area sensing and surveillance. The ability to network *anywhere, anytime* enables teleconferencing, home networking, sensor networks, personal area networks, and embedded computing applications [13].

Related Work

The most common way to establish communication is to form paths of intermediate nodes that lie within one another's transmission range and can directly communicate with each other. The mobile nodes act as hosts and routers at the same time in order to propagate packets along these paths. This approach of maintaining a global structure with respect to the temporary network is a difficult problem. Since nodes are moving, the underlying communication graph is changing, and the nodes have to adapt quickly to such changes and reestablish their routes. Busch and Tirthapura [2] provide the first analysis of the performance of some characteristic protocols [8,13] and show that in some cases they require $\Omega(u^2)$ time, where u is the number of nodes, to stabilize, i. e. be able to provide communication.

The work of Chatzigiannakis, Nikoletseas and Spirakis [5] focuses on networks where topological connectivity is subject to frequent, unpredictable change and studies the problem of efficient data delivery in sparse networks where network partitions can last for a significant period of time. In such cases, it is possible to have a small team of fast moving and versatile vehicles, to implement the support. These vehicles can be cars, motorcycles, helicopters or a collection of independently controlled mobile modules, i. e. robots. This specific approach is inspired by the work of Walter, Welch and Amato [14] that study the problem of motion co-ordination in distributed systems consisting of such robots, which can connect, disconnect and move around.

The use of mobility to improve performance in ad hoc mobile networks has been considered in different contexts in [6,9,11,15]. The primary objective has been to provide intermittent connectivity in a disconnected ad hoc network. Each solution achieves certain properties of end-to-end connectivity, such as delay and message loss among the nodes of the network. Some of them require long-range wireless transmission, other require that all nodes move pro-actively under the control of the protocol and collaborate so that they meet more often. The *key idea* of forcing only a subset of the nodes to facilitate communication is used in a similar way in [10,15]. However, [15] focuses in cases where only one node is available. Recently, the application of mobility to the domain of wireless sensor networks has been addressed in [3,10,12].

Open Problems

A number of problems related to the work of Chatzigiannakis, Nikoletseas and Spirakis [5] remain open. It is clear that the size of the support, k, the shape and the way the support moves affects the performance of end-to-end connectivity. An open issue is to investigate alternative structures for the support, different motion coordination strategies and comparatively study the corresponding effects on communication times. To this end, the support idea is extended to hierarchical and highly changing motion graphs in [4]. The idea of cooperative routing based on the existence of support nodes may also improve security and trust.

An important issue for the case where the network is sparsely populated or where the rate of motion is too high is to study the performance of path construction and maintenance protocols. Some work has be done in this direction in [2] that can be also used to investigate the end-to-end communication in wireless sensor networks. It is still unknown if there exist impossibility results for distributed algorithms that attempt to maintain structural information of the implied fragile network of virtual links.

Another open research area is to analyze the properties of end-to-end communication given certain support motion strategies. There are cases where the mobile nodes interactions may behave in a similar way to the Physics paradigm of *interacting particles* and their modeling. Studies of interaction times and propagation times in various graphs are reported in [7] and are still important to further research in this direction.

Experimental Results

In [5] an experimental evaluation is conducted via simulation in order to model the different possible situations regarding the geographical area covered by an ad-hoc mobile network. A number of experiments were carried out for grid-graphs (2D, 3D), random graphs ($G_{n,p}$ model), bipartite multi-stage graphs and two-level motion graphs. All results verify the theoretical analysis and provide useful insight on how to further exploit the support idea. In [4] the model of hierarchical and highly changing ad-hoc networks is investigated. The experiments indicate that, the pattern of the "snake" algorithm's performance remains the same even in such type of networks.

URL to Code

http://ru1.cti.gr

Cross References

► Mobile Agents and Exploration

Recommended Reading

1. Aldous, D., Fill, J.: Reversible markov chains and random walks on graphs. http://stat-www.berkeley.edu/users/aldous/book.html (1999). Accessed 1999
2. Busch, C., Tirthapura, S.: Analysis of link reversal routing algorithms. SIAM J. Comput. 35(2):305–326 (2005)
3. Chatzigiannakis, I., Kinalis, A., Nikoletseas, S.: Sink mobility protocols for data collection in wireless sensor networks. In: Zomaya, A.Y., Bononi, L. (eds.) 4th International Mobility and Wireless Access Workshop (MOBIWAC 2006), Terromolinos, pp 52–59
4. Chatzigiannakis, I., Nikoletseas, S.: Design and analysis of an efficient communication strategy for hierarchical and highly changing ad-hoc mobile networks. J. Mobile Netw. Appl. **9**(4), 319–332 (2004). Special Issue on Parallel Processing Issues in Mobile Computing
5. Chatzigiannakis, I., Nikoletseas, S., Spirakis, P.: Distributed communication algorithms for ad hoc mobile networks. J. Parallel Distrib. Comput. (JPDC) **63**(1), 58–74 (2003). Special Issue on Wireless and Mobile Ad-hoc Networking and Computing, edited by Boukerche A
6. Diggavi, S.N., Grossglauser, M., Tse, D.N.C.: Even one-dimensional mobility increases the capacity of wireless networks. IEEE Trans. Inf. Theory **51**(11), 3947–3954 (2005)
7. Dimitriou, T., Nikoletseas, S.E., Spirakis, P.G.: Analysis of the information propagation time among mobile hosts. In: Nikolaidis, I., Barbeau, M., Kranakis, E. (eds.) 3rd International Conference on Ad-Hoc, Mobile, and Wireless Networks (ADHOC-NOW 2004), pp 122–134. Lecture Notes in Computer Science (LNCS), vol. 3158. Springer, Berlin (2004)
8. Gafni, E., Bertsekas, D.P.: Distributed algorithms for generating loop-free routes in networks with frequently changing topology. IEEE Trans. Commun. **29**(1), 11–18 (1981)
9. Grossglauser, M., Tse, D.N.C.: Mobility increases the capacity of ad hoc wireless networks. IEEE/ACM Trans. Netw. **10**(4), 477–486 (2002)
10. Jain, S., Shah, R., Brunette, W., Borriello, G., Roy, S.: Exploiting mobility for energy efficient data collection in wireless sensor networks. J. Mobile Netw. Appl. **11**(3), 327–339 (2006)
11. Li, Q., Rus, D.: Communication in disconnected ad hoc networks using message relay. Journal of Parallel and Distributed Computing (JPDC) **63**(1), 75–86 (2003). Special Issue on Wire-

less and Mobile Ad-hoc Networking and Computing, edited by A Boukerche
12. Luo, J., Panchard, J., Piórkowski, M., Grossglauser, M., Hubaux, J.P.: Mobiroute: Routing towards a mobile sink for improving lifetime in sensor networks. In: Gibbons, P.B., Abdelzaher, T., Aspnes, J., Rao, R. (eds.) 2nd IEEE/ACM International Conference on Distributed Computing in Sensor Systems (DCOSS 2005). Lecture Notes in Computer Science (LNCS), vol. 4026, pp 480–497. Springer, Berlin (2006)
13. Perkins, C.E.: Ad Hoc Networking. Addison-Wesley, Boston (2001)
14. Walter, J.E., Welch, J.L., Amato, N.M.: Distributed reconfiguration of metamorphic robot chains. J. Distrib. Comput. **17**(2), 171–189 (2004)
15. Zhao, W., Ammar, M., Zegura, E.: A message ferrying approach for data delivery in sparse mobile ad hoc networks. In: Murai, J., Perkins, C., Tassiulas, L. (eds.) 5th ACM international symposium on Mobile ad hoc networking and computing (MobiHoc 2004), pp 187–198. ACM Press, Roppongi Hills, Tokyo (2004)

Competitive Auction

2001; Goldberg, Hartline, Wright
2002; Fiat, Goldberg, Hartline, Karlin

TIAN-MING BU
Department of Computer Science and Engineering,
Fudan University, Shanghai, China

Problem Definition

This problem studies the *one round*, *sealed-bid* auction model where an auctioneer would like to sell an idiosyncratic commodity with unlimited copies to n bidders and each bidder $i \in \{1, \dots, n\}$ will get at most one item.

First, for any i, bidder i bids a value b_i representing the price he is willing to pay for the item. They submit the bids simultaneously. After receiving the bidding vector $\mathbf{b} = (b_1, \dots, b_n)$, the auctioneer computes and outputs the allocation vector $\mathbf{x} = (x_1, \dots, x_n) \in \{0, 1\}^n$ and the price vector $\mathbf{p} = (p_1, \dots, p_n)$. If for any i, $x_i = 1$, then bidder i gets the item and pays p_i for it. Otherwise, bidder i loses and pays nothing. In the auction, the auctioneer's revenue is $\sum_{i=1}^{n} \mathbf{xp}^{\mathrm{T}}$.

Definition 1 (Optimal Single Price Omniscient Auction $\mathcal{F}$) Given a bidding vector $\mathbf{b}$ sorted in decreasing order,

$$\mathcal{F}(\mathbf{b}) = \max_{1 \le i \le n} i \cdot b_i .$$

Further,

$$\mathcal{F}^{(m)}(\mathbf{b}) = \max_{m \le i \le n} i \cdot b_i .$$

Obviously, $\mathcal{F}$ maximizes the auctioneer's revenue if only uniform price is allowed.

However, in this problem each bidder i is associated with a private value v_i representing the item's value in his opinion. So if bidder i gets the item, his payoff should be $v_i - p_i$. Otherwise, his payoff is 0. So for any bidder i, his payoff function can be formulated as $(v_i - p_i)x_i$. Furthermore, free will is allowed in the model. In other words, each bidder would bid some b_i different from his true value v_i, to maximize his payoff.

The objective of the problem is to design a *truthful* auction which could still maximize the auctioneer's revenue. An auction is *truthful* if for every bidder i, bidding his true value would maximize his payoff, regardless of the bids submitted by the other bidders [11,12].

Definition 2 (Competitive Auctions)
INPUT: the submitted bidding vector $\mathbf{b}$.
OUTPUT: the allocation vector $\mathbf{x}$ and the price vector $\mathbf{p}$.
CONSTRAINTS:
(a) Truthful
(b) The auctioneer's revenue is within a constant factor of the optimal single pricing for all inputs.

Key Results

Let $\mathbf{b}_{-i} = (b_1, \dots, b_{i-1}, b_{i+1}, \dots, b_n)$. f is any function from $\mathbf{b}_{-i}$ to the price.

```
1: for $i = 1$ to $n$ do
2:    if $f(\mathbf{b}_{-i}) \le b_i$ then
3:       $x_i = 1$ and $p_i = f(\mathbf{b}_i)$
4:    else
5:       $x_i = 0$
6:    end if
7: end for
```

Competitive Auction, Algorithm 1
Bid-independent Auction: $\mathcal{A}_f(\mathbf{b})$

Theorem 1 ([6]) *An auction is truthful if and only if it is equivalent to a bid-independent auction.*

Definition 3 A truthful auction $\mathcal{A}$ is β-competitive against $\mathcal{F}^{(m)}$ if for all bidding vectors $\mathbf{b}$, the expected profit of $\mathcal{A}$ on $\mathbf{b}$ satisfies

$$\mathbf{E}(\mathcal{A}(\mathbf{b})) \ge \frac{\mathcal{F}^{(m)}(\mathbf{b})}{\beta} .$$

Definition 4 (CostShare$_C$) ([10]) Given bids $\mathbf{b}$, this mechanism finds the largest k such that the highest k bidders' bids are at least C/k. Charge each of such k bidders C/k.

```
1: Partition bidding vector $\mathbf{b}$ uniformly at random into
       two sets $\mathbf{b}'$ and $\mathbf{b}''$.
2: Computer $\mathcal{F}' = \mathcal{F}(\mathbf{b}')$ and $\mathcal{F}'' = \mathcal{F}(\mathbf{b}'')$.
3: Running CostShare$_{\mathcal{F}''}$ on $\mathbf{b}'$ and CostShare$_{\mathcal{F}'}$ on $\mathbf{b}''$.
```

Competitive Auction, Algorithm 2
Sampling Cost Sharing Auction (SCS)

Theorem 2 ([6]) *SCS is 4-competitive against $\mathcal{F}^{(2)}$, and the bound is tight.*

Theorem 3 ([9]) *Let $\mathcal{A}$ be any truthful randomized auction. There exists an input bidding vector* $\mathbf{b}$ *on which* $E(\mathcal{A}(\mathbf{b})) \le \frac{\mathcal{F}^{(2)}(\mathbf{b})}{2.42}$.

Applications

As the Internet becomes more popular, more and more auctions are beginning to appear. Further, the items on sale in the auctions vary from antiques, paintings to digital goods such as mp3, licenses and network resources. Truthful auctions can reduce the bidders' cost of investigating the competitors' strategies, since truthful auctions encourage bidders to bid their true values. On the other hand, competitive auctions can also guarantee the auctioneer's profit. So this problem is very practical and significant. Over the last two years, designing and analyzing competitive auctions under various auction models have become a hot topic [1,2,3,4,5,7,8].

Cross References

▶ CPU Time Pricing
▶ Multiple Unit Auctions with Budget Constraint

Recommended Reading

1. Abrams, Z.: Revenue maximization when bidders have budgets. In: Proceedings of the seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA-06), Miami, FL, 22–26 January 2006, pp. 1074–1082. ACM Press, New York (2006)
2. Bar-Yossef, Z., Hildrum, K., Wu, F.: Incentive-compatible online auctions for digital goods. In: Proceedings of the 13th Annual ACM-SIAM Symposium On Discrete Mathematics (SODA-02), New York, 6–8 January 2002, pp. 964–970. ACM Press, New York (2002)
3. Borgs, C., Chayes, J.T., Immorlica, N., Mahdian, M., Saberi, A.: Multi-unit auctions with budget-constrained bidders. In: ACM Conference on Electronic Commerce (EC-05), 2005, pp. 44–51
4. Bu, T.-M., Qi, Q., Sun, A.W.: Unconditional competitive auctions with copy and budget constraints. In: Spirakis, P.G., Mavronicolas, M., Kontogiannis, S.C. (eds.) Internet and Network Economics, 2nd International Workshop, WINE 2006, Patras, Greece, 15–17 Dec 2006. Lecture Notes in Computer Science, vol. 4286, pp. 16–26. Springer, Berlin (2006)
5. Deshmukh, K., Goldberg, A.V., Hartline, J.D., Karlin, A.R.: Truthful and competitive double auctions. In: Möhring, R.H., Raman, R. (eds.) Algorithms–ESA 2002, 10th Annual European Symposium, Rome, Italy, 17–21 Sept 2002. Lecture Notes in Computer Science, vol. 2461, pp. 361–373. Springer, Berlin (2002)
6. Fiat, A., Goldberg, A.V., Hartline, J.D., Karlin, A.R.: Competitive generalized auctions. In: Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC-02), New York, 19–21 May 2002, pp. 72–81. ACM Press, New York (2002)
7. Goldberg, A.V., Hartline, J.D.: Competitive auctions for multiple digital goods. In: auf der Heide, F.M. (ed.) Algorithms – ESA 2001, 9th Annual European Symposium, Aarhus, Denmark, 28–31 Aug 2001. Lecture Notes in Computer Science, vol. 2161, pp. 416–427. Springer, Berlin (2001)
8. Goldberg, A.V. Hartline, J.D.: Envy-free auctions for digital goods. In: Proceedings of the 4th ACM Conference on Electronic Commerce (EC-03), New York, 9–12 June 2003, pp. 29–35. ACM Press, New York (2003)
9. Goldberg, A.V., Hartline, J.D., Wright, A.: Competitive auctions and digital goods. In: Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA-01), New York, 7–9 January 2001, pp. 735–744. ACM Press, New York (2001)
10. Moulin, H.: Incremental cost sharing: Characterization by coalition strategy-proofness. Social Choice and Welfare, **16**, 279–320 (1999)
11. Nisan, N.and Ronen, A.: Algorithmic mechanism design. In: Proceedings of the 31st Annual ACM Symposium on Theory of Computing (STOC-99), New York, May 1999, pp. 129–140. Association for Computing Machinery, New York (1999)
12. Parkes, D.C.: Chapter 2: Iterative Combinatorial Auctions. Ph. D. thesis, University of Pennsylvania (2004)

Complexity of Bimatrix Nash Equilibria

2006; Chen, Deng

Xi Chen[1], Xiaotie Deng[2]

[1] Computer Science and Technology, Tsinghua University, Beijing, China

[2] Department of Computer Science, City University of Hong Kong, Hong Kong, China

Keywords and Synonyms

Two-player nash; Two-player game; Two-person game; Bimatrix game

Problem Definition

In the middle of the last century, Nash [8] studied general non-cooperative games and proved that there exists a set of mixed strategies, now commonly referred to as a Nash equilibrium, one for each player, such that no player can benefit if it changes its own strategy unilaterally. Since the development of Nash's theorem, researchers have worked on how to compute Nash equilibria efficiently. Despite much effort in the last half century, no significant progress has been made on characterizing its algorithmic complexity, though both hardness results and algorithms have been developed for various modified versions.

An exciting breakthrough, which shows that computing Nash equilibria is possibly hard, was made by Daskalakis, Goldberg, and Papadimitriou [4], for games among four players or more. The problem was proven to be complete in **PPAD** (polynomial parity argument, directed version), a complexity class introduced by Papadimitriou in [9]. The work of [4] is based on the techniques developed in [6]. This hardness result was then improved to the three-player case by Chen and Deng [1], Daskalakis and Papadimitriou [5], independently, and with different proofs. Finally, Chen and Deng [2] proved that NASH, the problem of finding a Nash equilibrium in a bimatrix game (or two-player game), is **PPAD**-complete.

A bimatrix game is a non-cooperative game between two players in which the players have m and n choices of actions (or pure strategies), respectively. Such a game can be specified by two $m \times n$ matrices $\mathbf{A} = (a_{i,j})$ and $\mathbf{B} = (b_{i,j})$. If the first player chooses action i and the second player chooses action j, then their payoffs are $a_{i,j}$ and $b_{i,j}$, respectively. A mixed strategy of a player is a probability distribution over its choices. Let $\mathbb{P}^n$ denote the set of all probability vectors in $\mathbb{R}^n$, i. e., non-negative vectors whose entries sum to 1. Nash's equilibrium theorem on non-cooperative games, when specialized to bimatrix games, states that, for every bimatrix game $\mathcal{G} = (\mathbf{A}, \mathbf{B})$, there exists a pair of mixed strategies $(\mathbf{x}^* \in \mathbb{P}^m, \mathbf{y}^* \in \mathbb{P}^n)$, called a Nash equilibrium, such that for all $\mathbf{x} \in \mathbb{P}^m$ and $\mathbf{y} \in \mathbb{P}^n$,

$$(\mathbf{x}^*)^{\mathrm{T}}\mathbf{A}\mathbf{y}^* \geq \mathbf{x}^T\mathbf{A}\mathbf{y}^* \quad \text{and} \quad (\mathbf{x}^*)^{\mathrm{T}}\mathbf{B}\mathbf{y}^* \geq (\mathbf{x}^*)^{\mathrm{T}}\mathbf{B}\mathbf{y}.$$

Computationally, one might settle with an approximate Nash equilibrium. Let $\mathbf{A}_i$ denote the ith row vector of $\mathbf{A}$, and $\mathbf{B}_i$ denote the ith column vector of $\mathbf{B}$. An ϵ-well-supported Nash equilibrium of game $(\mathbf{A}, \mathbf{B})$ is a pair of mixed strategies $(\mathbf{x}^*, \mathbf{y}^*)$ such that,

$$\mathbf{A}_i\mathbf{y}^* > \mathbf{A}_j\mathbf{y}^* + \epsilon \Longrightarrow x_j^* = 0,\ \forall\ i,j : 1 \leq i,j \leq m;$$
$$(\mathbf{x}^*)^{\mathrm{T}}\mathbf{B}_i > (\mathbf{x}^*)^{\mathrm{T}}\mathbf{B}_j + \epsilon \Longrightarrow y_j^* = 0,\ \forall\ i,j : 1 \leq i,j \leq n.$$

Definition 1 (2-NASH and NASH) The input instance of problem 2-NASH is a pair $(\mathcal{G}, 0^k)$ where $\mathcal{G}$ is a bimatrix game, and the output is a 2^{-k}-well-supported Nash equilibrium of $\mathcal{G}$. The input of problem NASH is a bimatrix game $\mathcal{G}$ and the output is an exact Nash equilibrium of $\mathcal{G}$.

Key Results

A binary relation $R \subset \{0,1\}^* \times \{0,1\}^*$ is *polynomially balanced* if there exists a polynomial p such that for all pairs $(x,y) \in R$, $|y| \leq p(|x|)$. It is a *polynomial-time computable relation* if for each pair (x,y), one can decide whether or not $(x,y) \in R$ in time polynomial in $|x| + |y|$. The **NP** search problem Q_R specified by R is defined as follows: Given $x \in \{0,1\}^*$, if there exists y such that $(x,y) \in R$, return y, otherwise, return a special string "no".

Relation R is *total* if for every $x \in \{0,1\}^*$, there exists a y such that $(x,y) \in R$. Following [7], let **TFNP** denote the class of all **NP** search problems specified by total relations. A search problem $Q_{R_1} \in$ **TFNP** is *polynomial-time reducible* to problem $Q_{R_2} \in$ **TFNP** if there exists a pair of polynomial-time computable functions (f,g) such that for every x of R_1, if y satisfies that $(f(x),y) \in R_2$, then $(x, g(y)) \in R_1$. Furthermore, Q_{R_1} and Q_{R_2} are polynomial-time equivalent if Q_{R_2} is also reducible to Q_{R_1}.

The complexity class **PPAD** is a sub-class of **TFNP**, containing all the search problems which are polynomial-time reducible to:

Definition 2 (Problem LEAFD) The input instance of LEAFD is a pair $(M, 0^n)$ where M defines a polynomial-time Turing machine satisfying:

1. for every $v \in \{0,1\}^n$, $M(v)$ is an ordered pair (u_1, u_2) with $u_1, u_2 \in \{0,1\}^n \cup \{\text{"no"}\}$;
2. $M(0^n) = (\text{"no"}, 1^n)$ and the first component of $M(1^n)$ is 0^n.

This instance defines a directed graph $G = (V, E)$ with $V = \{0,1\}^n$. Edge $(u,v) \in E$ iff v is the second component of $M(u)$ and u is the first component of $M(v)$.

The output of problem LEAFD is a directed leaf of G other than 0^n. Here a vertex is called a *directed leaf* if its out-degree plus in-degree equals one.

A search problem in **PPAD** is said to be *complete* in **PPAD** (or **PPAD**-complete), if there exists a polynomial-time reduction from LEAFD to it.

Theorem ([2]) 2-Nash *and* Nash *are* **PPAD**-*complete.*

Applications

The concept of Nash equilibria has traditionally been one of the most influential tools in the study of many disciplines involved with strategies, such as political science and economic theory. The rise of the Internet and the study of its anarchical environment have made the Nash equilibrium an indispensable part of computer science. Over the past decades, the computer science community have contributed a lot to the design of efficient algorithms for related problems. This sequence of results [1,2,3,4,5,6], for the first time, provide *some evidence* that the problem of finding a Nash equilibrium is possibly hard for **P**. These results are very important to the emerging discipline, Algorithmic Game Theory.

Open Problems

This sequence of works show that $(r+1)$-player games are polynomial-time reducible to r-player games for every $r \geq 2$, but the reduction is carried out by first reducing $(r+1)$-player games to a fixed point problem, and then further to r-player games. Is there a natural reduction that goes directly from $(r+1)$-player games to r-player games? Such a reduction could provide a better understanding for the behavior of multi-player games.

Although many people believe that **PPAD** is hard for **P**, there is no strong evidence for this belief or intuition. The natural open problem is: Can one rigorously prove that class **PPAD** is hard, under one of those generally believed assumptions in theoretical computer science, like "**NP** is not in **P**" or "one way function exists"? Such a result would be extremely important to both Computational Complexity Theory and Algorithmic Game Theory.

Cross References

► General Equilibrium
► Leontief Economy Equilibrium
► Non-approximability of Bimatrix Nash Equilibria

Recommended Reading

1. Chen, X., Deng, X.: 3-Nash is ppad-complete. ECCC, TR05–134 (2005)
2. Chen, X., Deng, X.: Settling the complexity of two-player Nash-equilibrium. In: FOCS'06, Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science, 2006, pp. 261–272
3. Chen, X., Deng, X., Teng, S.H.: Computing Nash equilibria: approximation and smoothed complexity. In: FOCS'06, Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science, 2006, pp. 603–612
4. Daskalakis, C., Goldberg, P.W. Papadimitriou, C.H.: The complexity of computing a Nash equilibrium. In: STOC'06, Proceedings of the 38th ACM Symposium on Theory of Computing, 2006, pp. 71–78
5. Daskalakis, C., Papadimitriou, C.H.: Three-player games are hard. ECCC, TR05–139 (2005)
6. Goldberg, P.W., Papadimitriou, C.H.: Reducibility among equilibrium problems. In: STOC'06, Proceedings of the 38th ACM Symposium on Theory of Computing, 2006, pp. 61–70
7. Megiddo, N., Papadimitriou, C.H.: On total functions, existence theorems and computational complexity. Theor. Comp. Sci. **81**, 317–324 (1991)
8. Nash, J.F.: Equilibrium point in n-person games. In: Proceedings of the National Academy of the USA, vol. 36, issue 1, pp. 48–49 (1950)
9. Papadimitriou, C.H.: On the complexity of the parity argument and other inefficient proofs of existence. J. Comp. Syst. Sci. **48**, 498–532 (1994)

Complexity of Core

2001; Fang, Zhu, Cai, Deng

QIZHI FANG
Department of Mathematics,
Ocean University of China,
Qingdao, China

Keywords and Synonyms

Balanced; Least-core

Problem Definition

The core is the most important solution concept in cooperative game theory, which is based on the coalition rationality condition: no subgroup of the players will do better if they break away from the joint decision of all players to form their own coalition. The principle behind this condition is very similar and can be seen as an extension to that of the Nash Equilibrium. The problem of determining the core of a cooperative game naturally brings in issues of algorithms and complexity. The work of Fang, Zhu, Cai, and Deng [4] discusses the computational complexity issues related to the cores of some cooperative game models, such as, flow games and Steiner tree games.

A cooperative game with side payments is given by the pair $(N, \boldsymbol{v})$, where $N = \{1, 2, \cdots, n\}$ is the player set and $\boldsymbol{v} : 2^N \to R$ is the characteristic function. For each coalition $S \subseteq N$, the value $v(S)$ is interpreted as the profit or cost achieved by the collective action of players in S without any assistance of players in $N \setminus S$. A game is called a profit (cost) game if $v(S)$ measures the profit (cost) achieved by the coalition S. Here, the definitions are only given for profit games, symmetric statements hold for cost games. A vector $\boldsymbol{x} = \{x_1, x_2, \cdots, x_n\}$ is called an imputation if it satisfies $\sum_{i \in N} x_i = v(N)$ and $\forall i \in N : x_i \geq v(\{i\})$. The core of the game $(N, \boldsymbol{v})$ is defined as:

$$C(v) = \{\boldsymbol{x} \in R^n : \boldsymbol{x}(N) = v(N) \text{ and } \boldsymbol{x}(S) \geq \boldsymbol{v}(S),\ \forall S \subseteq N\},$$

where $x(S) = \sum_{i \in S} x_i$ for $S \subseteq N$. A game is called *balanced*, if its core is non-empty; and *totally balanced*, if every subgame (i. e., the game obtained by restricting the player set to a coalition and the characteristic function to the power set of that coalition) is balanced.

It is a challenge for the algorithmic study of the core, since there are an exponential number of constraints imposed on its definition. The following computational complexity questions have attracted much attention from researchers:

(1)*Testing balancedness:* Can it be tested in polynomial time whether a given instance of the game has a non-empty core?

(2)*Checking membership:* Can it be checked in polynomial time whether a given imputation belongs to the core?

(3)*Finding a core member:* Is it possible to find an imputation in the core in polynomial time?

In reality, however, there is an important case in which the characteristic function value of a coalition can usually be evaluated via a combinatorial optimization problem, subject to constraints of resources controlled by the players of this coalition. In such circumstances, the input size of a game is the same as that of the related optimization problem, which is usually polynomial in the number of players. Therefore, this class of games, called combinatorial optimization games, fits well into the framework of algorithm theory. Flow games and Steiner tree games discussed in Fang et al. [4] fall within this scope.

FLOW GAME Let $D = (V, E; \omega; s, t)$ be a directed flow network, where V is the vertex set, E is the arc set, $\omega\colon E \to R^+$ is the arc capacity function, s and t are the source and the sink of the network, respectively. Assume that each player controls one arc in the network. The value of a maximum flow can be viewed as the profit achieved by the players in cooperation. Then the flow game $\Gamma_f = (E, \boldsymbol{v})$ associated with the network D is defined as follows:

(i) The player set is E;

(ii) $\forall S \subseteq E$, $\boldsymbol{v}(S)$ is the value of a maximum flow from s to t in the subnetwork of D consisting only of arcs belonging to S.

In Kailai and Zemel [6] and Deng et al. [2], it was shown that the flow game is totally balanced and finding a core member can be done in polynomial time.

Problem 1 *(Checking membership for flow game)*
INSTANCE: A flow network $D = (V, E; \omega; s, t)$ and $\boldsymbol{x} : E \to R^+$.
QUESTION: Is it true that $\boldsymbol{x}(E) = \boldsymbol{v}(E)$ and $\boldsymbol{x}(S) \geq \boldsymbol{v}(S)$ for all subsets $S \subset E$?

STEINER TREE GAME Let $G = (V, E; \omega)$ be an edge-weighted graph with $V = \{v_0\} \cup N \cup M$, where $N, M \subseteq V \setminus \{v_0\}$ are disjoint. v_0 represents a central supplier, N represents the consumer set, M represents the switch set, and $\omega(e)$ denotes the cost of connecting the two endpoints of edge e directly. It is required to connect all the consumers in N to the central supplier v_0. The connection is not limited to using direct links between two consumers or a consumer and the central supplier, it may pass through some switches in M. The aim is to construct the cheapest connection and distribute the connection cost among the consumers fairly. Then the associated Steiner tree game $\Gamma_s = (N, \gamma)$ is defined as follows:

(i) The player set is N;

(ii) $\forall\ S \subseteq N$, $\gamma(S)$ is the weight of a minimum Steiner tree on G w.r.t. the set $S \cup \{v_0\}$, that is, $\gamma(S) = \min\{\sum_{e \in E_S} \omega(e) : T_S = (V_S, E_S)$ is a subtree of G with $V_S \supseteq S \cup \{v_0\}\}$.

Different from flow games, the core of a Steiner tree game may be empty. An example with an empty core was given in Megiddo [9].

Problem 2 *(Testing balancedness for a Steiner tree game)*
INSTANCE: An edge-weighted graph $G = (V, E; \omega)$ with $V = \{v_0\} \cup N \cup M$.
QUESTION: Does there exist a vector $\boldsymbol{x} : N \to R^+$ such that $\boldsymbol{x}(N) = \gamma(N)$ and $\boldsymbol{x}(S) \leq \gamma(S)$ for all subsets $S \subset N$?

Problem 3 *(Checking membership for a Steiner tree game)*
INSTANCE: An edge-weighted graph $G = (V, E; \omega)$ with $V = \{v_0\} \cup N \cup M$ and $\boldsymbol{x} : N \to R^+$.
QUESTION: Is it true that $\boldsymbol{x}(N) = \gamma(N)$ and $\boldsymbol{x}(S) \leq \gamma(S)$ for all subsets $S \subset N$?

Key Results

Theorem 1 *It is $\mathcal{NP}$-complete to show that, given a flow game $\Gamma_f = (E, \boldsymbol{v})$ defined on network $D = (V, E; \omega; s, t)$ and a vector $\boldsymbol{x}\colon E \to R^+$ with $\boldsymbol{x}(E) = \boldsymbol{v}(E)$, whether there exists a coalition $S \subset N$ such that $\boldsymbol{x}(S) < \boldsymbol{v}(S)$. That is, checking membership of the core for flow games is co-$\mathcal{NP}$-complete.*

The proof of Theorem 1 yields directly the same conclusion for linear production games. In Owen's linear production game [10], each player j ($j \in N$) is in possession of an individual resource vector $\boldsymbol{b}^j$. For a coalition S of players, the profit obtained by S is the optimum value of the following linear program:

$$\max\{c^t y : Ay \le \sum_{j \in S} \boldsymbol{b}^j, \ y \ge 0\}.$$

That is, the characteristic function value is what the coalition can achieve in the linear production model with the resources under their control. Owen showed that one imputation in the core can also be constructed through an optimal dual solution to the linear program which determines the value of N. However, there are in general some imputations in the core which cannot be obtained in this way.

Theorem 2 *Checking membership of the core for linear production games is co-$\mathcal{NP}$-complete.*

The problem of finding a minimum Steiner tree in a network is $\mathcal{NP}$-hard, therefore, in a Steiner tree game, the value $\gamma(S)$ of each coalition S may not be obtained in polynomial time. It implies that the complement problem of checking membership of the core for Steiner tree games may not be in $\mathcal{NP}$.

Theorem 3 *It is $\mathcal{NP}$-hard to show that, given a Steiner tree game $\Gamma_s = (N, \gamma)$ defined on network $G = (V, E; \omega)$ and a vector $\boldsymbol{x} : N \to R^+$ with $\boldsymbol{x}(N) = \gamma(N)$, whether there exists a coalition $S \subset N$ such that $\boldsymbol{x}(S) > \gamma(S)$. That is, checking membership of the core for Steiner tree games is $\mathcal{NP}$-hard.*

Theorem 4 *Testing balancedness for Steiner tree games is $\mathcal{NP}$-hard.*

Given a Steiner tree game $\Gamma_s = (N, \gamma)$ defined on network $G = (V, E; \omega)$ and a subset $S \subseteq N$, in the subgame (S, γ_S), the value $\gamma(S')$ $(S' \subseteq S)$ is the weight of a minimum Steiner tree of G w.r.t. the subset $S' \cup \{v_0\}$, where all the vertices in $N \setminus S$ are treated as switches but not consumers. It is further proved in Fang et al. [4] that determining whether a Steiner tree game is totally balanced is also $\mathcal{NP}$-hard. This is the first example of $\mathcal{NP}$-hardness for the totally balanced condition.

Theorem 5 *Testing total balancedness for Steiner tree games is $\mathcal{NP}$-hard.*

Applications

The computational complexity results on the cores of combinatorial optimization games have been as diverse as the corresponding combinatorial optimization problems. For example:

(1) In matching games [1], testing balancedness, checking membership, and finding a core member can all be done in polynomial time;

(2) In flow games and minimum-cost spanning tree games [3,4], although their cores are always non-empty and a core member can be found in polynomial time, the problem of checking membership is *co-$\mathcal{NP}$*-complete;

(3) In facility location games [5], the problem of testing balancedness is in general $\mathcal{NP}$-hard, however, given the information that the core is non-empty, both finding a core member and checking membership can be solved efficiently;

(4) In a game of sum of edge weight defined on a graph [2], all the problems of testing balancedness, checking membership, and finding a core member are $\mathcal{NP}$-hard.

To make the study of complexity and algorithms for cooperative games meaningful to corresponding application areas, it is suggested that computational complexity be taken as an important factor in considering rationality and fairness of a solution concept, in a way derived from the concept of bounded rationality [3,8]. That is, the players are not willing to spend super-polynomial time to search for the most suitable solution. In the case when the solutions of a game do not exist or are difficult to compute or check, it may not be simple to dismiss the problem as hopeless, especially when the game arises from important applications. Hence, various conceptual approaches are proposed to resolve this problem.

When the core of a game is empty, it motivates conditions ensuring non-emptiness of approximate cores. A natural way to approximate the core is the *least core*. Let $(N, \boldsymbol{v})$ be a profit cooperative game. Given a real number ε, the ε-core is defined to contain the allocations such that $\boldsymbol{x}(S) \ge \boldsymbol{v}(S) - \varepsilon$ for each non-empty proper subset S of N. The *least core* is the intersection of all non-empty ε-cores. Let ε^* be the minimum value of ε such that the ε-core is empty, then the least core is the same as the ε^*-core.

The concept of the least core poses new challenges in regard to algorithmic issues. The most natural problem is how to efficiently compute the value ε^* for a given cooperative game. The catch is that the computation of ε^* requires solving of a linear program with an exponential number of constrains. Though there are cases where this value can be computed in polynomial time [7], it is in general very hard. If the value of ε^* is considered to represent some subsidies given by the central authority to ensure the existence of the cooperation, then it is significant to give the approximate value of it even when its computation is $\mathcal{NP}$-hard.

Another possible approach is to interpret approximation as bounded rationality. For example, it would be interesting to know if there is any game with a property that for any $\varepsilon > 0$, checking membership in the ε-core can be done in polynomial time but it is $\mathcal{NP}$-hard to tell if an imputation is in the core. In such cases, the restoration of cooperation would be a result of bounded rationality. That is to say, the players would not care an extra gain or loss of ε as the expense of another order of degree of computational resources. This methodology may be further applied to other solution concepts.

Cross References

▶ General Equilibrium
▶ Nucleolus
▶ Routing

Recommended Reading

1. Deng, X., Ibaraki, T., Nagamochi, H.: Algorithmic Aspects of the Core of Combinatorial Optimization Games. Math. Oper. Res. **24**, 751–766 (1999)
2. Deng, X., Papadimitriou, C.: On the Complexity of Cooperative Game Solution Concepts. Math. Oper. Res. **19**, 257–266 (1994)
3. Faigle, U., Fekete, S., Hochstättler, W., Kern, W.: On the Complexity of Testing Membership in the Core of Min-Cost Spanning Tree Games. Int. J. Game. Theor. **26**, 361–366 (1997)
4. Fang, Q., Zhu, S., Cai, M., Deng, X.: Membership for core of LP games and other games. COCOON 2001 Lecture Notes in Computer Science, vol. 2108, pp 247–246. Springer-Verlag, Berlin Heidelberg (2001)
5. Goemans, M.X., Skutella, M.: Cooperative Facility Location Games. J. Algorithms **50**, 194–214 (2004)
6. Kalai, E., Zemel, E.: Generalized Network Problems Yielding Totally Balanced Games. Oper. Res. **30**, 998–1008 (1982)
7. Kern, W., Paulusma, D.: Matching Games: The Least Core and the Nucleolus. Math. Oper. Res. **28**, 294–308 (2003)
8. Megiddo, N.: Computational Complexity and the Game Theory Approach to Cost Allocation for a Tree. Math. Oper. Res. **3**, 189–196 (1978)
9. Megiddo, N.: Cost Allocation for Steiner Trees. Netw. **8**, 1–6 (1978)
10. Owen, G.: On the Core of Linear Production Games. Math. Program. **9**, 358–370 (1975)

Compressed Pattern Matching

2003; Kida, Matsumoto, Shibata, Takeda, Shinohara, Arikawa

Masayuki Takeda
Department of Informatics, Kyushu University, Fukuoka, Japan

Keywords and Synonyms

String matching over compressed text; Compressed string search

Problem Definition

Let **c** be a given compression algorithm, and let $\mathbf{c}(A)$ denote the result of **c** compressing a string A. Given a pattern string P and a compressed text string $\mathbf{c}(T)$, the *compressed pattern matching (CPM)* problem is to find all occurrences of P in T without decompressing T. The goal is to perform the task in less time compared with a decompression followed by a simple search, which takes $O(|P| + |T|)$ time (assuming $O(|T|)$ time is enough for decompression). A CPM algorithm is said to be *optimal* if it runs in $O(|P| + |\mathbf{c}(T)|)$ time. The CPM problem was first defined in the work of Amir and Benson [1], and many studies have been made over different compression formats.

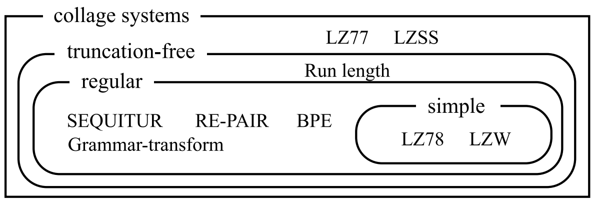
Collage Systems

Collage systems are useful CPM-oriented abstractions of compression formats, introduced by Kida et al. [9]. Algorithms designed for collage systems can be implemented for many different compression formats. In the same paper they designed a general Knuth–Morris–Pratt (KMP) algorithm for collage systems. A general Boyer–Moore (BM) algorithm for collage systems was also designed by almost the same authors [18].

A *collage system* is a pair $\langle \mathcal{D}, \mathcal{S} \rangle$ defined as follows. $\mathcal{D}$ is a sequence of assignments $X_1 = expr_1; X_2 = expr_2; \ldots; X_n = expr_n$, where, for each $k = 1, \ldots, n$, X_k is a variable and $expr_k$ is any of the form:

$$a \text{ for } a \in \Sigma \cup \{\varepsilon\}, \qquad \text{(primitive assignment)}$$
$$X_i X_j \text{ for } i, j < k, \qquad \text{(concatenation)}$$
$$^{[j]}X_i \text{ for } i < k \text{ and a positive integer } j, \qquad (j \text{ length prefix truncation})$$
$$X_i^{[j]} \text{ for } i < k \text{ and a positive integer } j, \qquad (j \text{ length suffix truncation})$$
$$(X_i)^j \text{ for } i < k \text{ and a positive integer } j. \qquad (j \text{ times repetition})$$

By *the j length prefix (resp. suffix) truncation* we mean an operation on strings which takes a string w and returns the string obtained from w by removing its prefix (resp. suffix) of length j. The variables X_k represent the strings $\overline{X_k}$ obtained by evaluating their expressions. The *size* of $\mathcal{D}$ is the number n of assignments and denoted by $|\mathcal{D}|$. Let *height*$(\mathcal{D})$ denote the maximum dependence in $\mathcal{D}$. $\mathcal{S}$ is a sequence $X_{i_1} \cdots X_{i_\ell}$ of variables defined in $\mathcal{D}$. The *length*



Compressed Pattern Matching, Figure 1
Hierarchy of collage systems

of S is the number ℓ of variables in S and denoted by $|S|$. It can thus be considered that $|\mathbf{c}(T)| = |\mathcal{D}| + |S|$.

A collage system $\langle \mathcal{D}, S \rangle$ represents the string obtained by concatenating the strings $\overline{X_{i_1}}, \ldots, \overline{X_{i_\ell}}$ represented by variables $X_{i_1}, \ldots, X_{i_\ell}$ of S. It should be noted that any collage system can be converted into the one with $|S| = 1$, by adding a series of assignments with concatenation operations into $\mathcal{D}$. This may imply S is unnecessary. However, a variety of compression schemes can be captured naturally by separating $\mathcal{D}$ (defining *phrases*) from S (giving a factorization of text T into phrases). How to express compressed texts for existing compression schemes is found in [9].

A collage system is said to be *truncation-free* if $\mathcal{D}$ contains no truncation operation, and *regular* if $\mathcal{D}$ contains neither repetition nor truncation operation. A regular collage system is *simple* if $|\overline{Y}| = 1$ or $|\overline{Z}| = 1$ for every assignment $X = YZ$. Figure 1 gives the hierarchy of collage systems. The collage systems for RE-PAIR, SEQUITUR, Byte-Pair-Encoding (BPE), and the grammar-transform based compression scheme are regular. In the Lempel–Ziv family, the collage systems for LZ78/LZW are simple, while those for LZ77/LZSS are not truncation-free.

Key Results

It is straightforward to design an optimal solution for run-length encoding. For the two-dimensional run-length encoding, used by FAX transmission, an optimal solution was given by Amir, Benson, and Farach [3].

Theorem 1 (Amir et al. [3]) *There exists an optimal solution to the CPM problem for two-dimensional run-length encoding scheme.*

The same authors showed in [2] an almost optimal solution for LZW compression.

Theorem 2 (Amir et al. [2]) *The first-occurrence version of the CPM problem for LZW can be solved in* $O(|P|^2 + |\mathbf{c}(T)|)$ *time and space.*

An extension of [2] to the multi-pattern matching (dictionary matching) problem was presented by Kida et al. [10], together with the first experimental results in this area.

For LZ77 compression scheme, Farach and Thorup [6] presented the following result.

Theorem 3 (Farach and Thorup [6]) *Given an LZ77 compressed string Z of a text T, and given a pattern P, there is a randomized algorithm to decide if P occurs in T which runs in* $O(|Z| \log^2(|T|/|Z|) + |P|)$ *time.*

Lempel–Ziv factorization is a version of LZ77 compression without *self-referencing*. The following relation is present between Lempel–Ziv factorizations and collage systems.

Theorem 4 (Gąsieniec et al. [7]; Rytter [16]) *The Lempel–Ziv factorization Z of T can be transformed into a collage system of size* $O(|Z| \cdot \log |Z|)$ *generating T in* $O(|Z| \cdot \log |Z|)$ *time, and into a regular collage system of size* $O(|Z| \cdot \log |T|)$ *generating T in* $O(|Z| \cdot \log |T|)$ *time.*

The result of Amir et al. [2] was generalized in the work of Kida et al. [9] via the unified framework of collage systems.

Theorem 5 (Kida et al. [9]) *The CPM problem for collage systems can be solved in* $O\big((|\mathcal{D}|+|S|)\cdot \mathit{height}(\mathcal{D})+|P|^2+\mathit{occ}\big)$ *time using* $O(|\mathcal{D}| + |P|^2)$ *space, where occ is the number of pattern occurrences. The factor* $\mathit{height}(\mathcal{D})$ *is dropped for truncation-free collage systems.*

The algorithm of [9] has two stages: First it preprocesses $\mathcal{D}$ and P, and second it processes the variables of S. In the second stage, it simulates the move of a KMP automaton running on uncompressed text, by using two functions *Jump* and *Output*. Both these functions take a state q and a variable X as input. The former is used to substitute just one state transition for the consecutive state transitions of the KMP automaton for the string $\overline{X}$ for each variable X of S. The latter is used to report all pattern occurrences found during the state transitions. Let δ be the state-transition function of the KMP automaton. Then $\mathit{Jump}(q, X) = \delta(q, \overline{X})$ and $\mathit{Output}(q, X)$ is the set of lengths $|w|$ of non-empty prefixes w of $\overline{X}$ such that $\delta(q, w)$ is the final state. A naive two-dimensional array implementation of the two functions requires $\Omega(|\mathcal{D}| \cdot |P|)$ space. The data structures of [9] use only $O(|\mathcal{D}| + |P|^2)$ space, are built in $O(|\mathcal{D}| \cdot \mathit{height}(\mathcal{D}) + |P|^2)$ time, and enable us to compute $\mathit{Jump}(q, X)$ in $O(1)$ time and enumerate the set $\mathit{Output}(q, X)$ in $O(\mathit{height}(\mathcal{D}) + \ell)$ time where $\ell = |\mathit{Output}(q, X)|$. The factor $\mathit{height}(\mathcal{D})$ is dropped for truncation-free collage systems.

Another criterion of CPM algorithms is focused on the amount of extra space [4]. A CPM algorithm is *inplace* if the amount of extra space is proportional to the input size of P.

Theorem 6 (Amir et al. [4]) *There exists an inplace CPM algorithm for a two-dimensional run-length encoding scheme which runs in $O(|\mathbf{c}(T)| + |P| \log \sigma)$ time using extra $O(\mathbf{c}(P))$ space, where σ is the minimum of $|P|$ and the alphabet size.*

Many variants of the CPM problem exist. In what follows, some of them are briefly sketched. *Fully-compressed pattern matching (FCPM)* is the complicated version where both T and P are given in a compressed format. A straight-line program is a regular collage system with $|S| = 1$.

Theorem 7 (Miyazaki et al. [13]) *The FCPM problem for straight-line programs is solved in $O(|\mathbf{c}(T)|^2 \cdot |\mathbf{c}(P)|^2)$ time using $O(|\mathbf{c}(T)| \cdot |\mathbf{c}(P)|)$ space.*

Approximate compressed pattern matching (ACPM) refers to the case where errors are allowed.

Theorem 8 (Kärkkäinen et al. [8]) *Under the Levenshtein distance model, the ACPM problem can be solved in $O(k \cdot |P| \cdot |\mathbf{c}(T)| + occ)$ time for LZ78/LZW, and in $O(|P| \cdot (k^2 \cdot |\mathcal{D}| + k \cdot |S|) + occ)$ time for regular collage systems, where k is the given error threshold.*

Theorem 9 (Makinen et al. [11]) *Under a weighted edit distance model, the ACPM problem for run-length encoding can be solved in $O(|P| \cdot |\mathbf{c}(P)| \cdot |\mathbf{c}(T)|)$ time.*

Regular expression compressed pattern matching (RCPM) refers to the case where P can be a regular expression.

Theorem 10 (Navarro [14]) *The RCPM problem can be solved in $O(2^{|P|} + |P| \cdot |\mathbf{c}(T)| + occ \cdot |P| \cdot \log |P|)$ time, where occ is the number of occurrences of P in T.*

Applications

CPM techniques enable us to search directly in compressed text databases. One interesting application is searching over compressed text databases on handheld devices, such as PDAs, in which memory, storage, and CPU power are limited.

Experimental Results

One important goal of the CPM problem is to perform a CPM task faster than a decompression followed by a simple search. Kida et al. [10] showed experimentally that their algorithms achieve the goal. Navarro and Tarhio [15] presented BM type algorithms for LZ78/LZW compression schemes, and showed they are twice as fast as a decompression followed by a search using the best algorithms. (The code is available at: www.dcc.uchile.cl/gnavarro/software.)

Another challenging goal is to perform a CPM task faster than a simple search over original files in the uncompressed format. The goal is achieved by Manber [12] (with his own compression scheme), and by Shibata et al. [17] (with BPE). Their search time reduction ratios are nearly the same as their compression ratios. Unfortunately the compression ratios are not very high. Moura et al. [5] achieved the goal by using a bytewise Huffman code on words. The compression ratio is relatively high, but only searching for whole words and phrases is allowed.

Cross References

▶ Multidimensional compressed pattern matching is the complex version of CPM where the text and the pattern are multidimensional strings in a compressed format. ▶ Sequential exact string matching, ▶ sequential approximate string matching, ▶ regular expression matching, respectively, refer to the simplified versions of CPM, ACPM, RCPM where the text and the pattern are given as uncompressed strings.

Recommended Reading

1. Amir, A., Benson, G.: Efficient two-dimensional compressed matching. In: Proc. Data Compression Conference '92 (DCC'92), pp. 279 (1992)
2. Amir, A., Benson, G., Farach, M.: Let sleeping files lie: Pattern matching in Z-compressed files. J. Comput. Syst. Sci. **52**(2), 299–307 (1996)
3. Amir, A., Benson, G., Farach, M.: Optimal two-dimensional compressed matching. J. Algorithms **24**(2), 354–379 (1997)
4. Amir, A., Landau, G.M., Sokol, D.: Inplace run-length 2d compressed search. Theor. Comput. Sci. **290**(3), 1361–1383 (2003)
5. de Moura, E., Navarro, G., Ziviani, N., Baeza-Yates, R.: Fast and flexible word searching on compressed text. ACM Trans. Inf. Syst. **18**(2), 113–139 (2000)
6. Farach, M., Thorup, M.: String-matching in Lempel–Ziv compressed strings. Algorithmica **20**(4), 388–404 (1998)
7. Gąsieniec, L., Karpinski, M., Plandowski, W., Rytter, W.: Efficient algorithms for Lempel–Ziv encoding. In: Proc. 5th Scandinavian Workshop on Algorithm Theory (SWAT'96). LNCS, vol. 1097, pp. 392–403 (1996)
8. Kärkkäinen, J., Navarro, G., Ukkonen, E.: Approximate string matching on Ziv–Lempel compressed text. J. Discret. Algorithms **1**(3–4), 313–338 (2003)
9. Kida, T., Matsumoto, T., Shibata, Y., Takeda, M., Shinohara, A., Arikawa, S.: Collage systems: a unifying framework for compressed pattern matching. Theor. Comput. Sci. **298**(1), 253–272 (2003)
10. Kida, T., Takeda, M., Shinohara, A., Miyazaki, M., Arikawa, S.: Multiple pattern matching in LZW compressed text. J. Discret. Algorithms **1**(1), 133–158 (2000)

11. Makinen, V., Navarro, G., Ukkonen, E.: Approximate matching of run-length compressed strings. Algorithmica **35**(4), 347–369 (2003)
12. Manber, U.: A text compression scheme that allows fast searching directly in the compressed file. ACM Trans. Inf. Syst. **15**(2), 124–136 (1997)
13. Miyazaki, M., Shinohara, A., Takeda, M.: An improved pattern matching algorithm for strings in terms of straight-line programs. J. Discret. Algorithms **1**(1), 187–204 (2000)
14. Navarro, G.: Regular expression searching on compressed text. J. Discret. Algorithms **1**(5–6), 423–443 (2003)
15. Navarro, G., Tarhio, J.: LZgrep: A Boyer–Moore string matching tool for Ziv–Lempel compressed text. Softw. Pract. Exp. **35**(12), 1107–1130 (2005)
16. Rytter, W.: Application of Lempel–Ziv factorization to the approximation of grammar-based compression. Theor. Comput. Sci. **302**(1–3), 211–222 (2003)
17. Shibata, Y., Kida, T., Fukamachi, S., Takeda, M., Shinohara, A., Shinohara, T., Arikawa, S.: Speeding up pattern matching by text compression. In: Proc. 4th Italian Conference on Algorithms and Complexity (CIAC'00). LNCS, vol. 1767, pp. 306–315. Springer, Heidelberg (2000)
18. Shibata, Y., Matsumoto, T., Takeda, M., Shinohara, A., Arikawa, S.: A Boyer–Moore type algorithm for compressed pattern matching. In: Proc. 11th Annual Symposium on Combinatorial Pattern Matching (CPM'00). LNCS, vol. 1848, pp. 181–194. Springer, Heidelberg (2000)

Compressed Suffix Array

2003; Grossi, Gupta, Vitter

VELI MÄKINEN
Department of Computer Science,
University of Helsinki, Helsinki, Finland

Keywords and Synonyms

Compressed full-text indexing; Compressed suffix tree

Problem Definition

Given a *text string* $T = t_1 t_2 \dots t_n$ over an alphabet Σ of size σ, the *compressed full-text indexing (CFTI)* problem asks to create a space-efficient data structure capable of efficiently simulating the functionalities of a *full-text index* build on T.

A simple example of a full-text index is *suffix array* $A[1, n]$ that contains a permutation of the interval $[1, n]$, such that $T[A[i], n] < T[A[i+1], n]$ for all $1 \le i < n$, where "<" between strings is the lexicographical order. Using suffix array, the occurrences of a given pattern $P = p_1 p_2 \dots p_m$ in T can be found using two binary searches in $O(m \log n)$ time.

The CFTI problem related to suffix arrays is easily stated; find a space-efficient data structure supporting the retrieval of value $A[i]$ for any i efficiently. Such a solution is called *compressed suffix array*. Usually compressed suffix arrays support, as well, retrieving of the inverse $A^{-1}[j] = i$ for any given j.

If the compressed full-text index functions without the text and contains enough information to retrieve any substring of T, then this index is called *self-index*, as it can be used as a representation of T. See the entry *Compressed Text Indexing* for another approach to self-indexing, and [9] for a comprehensive survey on the topic.

The CFTI problem can be stated on any full-text index, as long as the set of operations the data structure should support is rigorously defined. For example, a *compressed suffix tree* should simulate all the operations of classical *suffix trees*.

The classical full-text indexes occupy $O(n \log n)$ bits, typically with large constant factors. The typical goals in CFTI can be characterized by the degree of ambition; find a structure whose space-requirement is:

(i) proportional to the text size, i. e. $O(n \log \sigma)$ bits;
(ii) asymptotically optimal in the text size, i. e. $n \log \sigma (1 + o(1))$ bits;
(iii) proportional to the *compressed* text size, i. e. $O(nH_k)$ bits, where H_k is the (empirical) *k-th order entropy* of T[1]; or even
(iv) asymptotically optimal in the compressed text size, i. e. $nH_k + o(n \log \sigma)$ bits.

Key Results

The first solution to the problem is by Grossi and Vitter [3] who exploit the regularities of suffix array via the Ψ-function:

Definition 1 Given suffix array $A[1, n]$, *function* $\Psi : [1, n] \to [1, n]$ is defined so that, for all $1 \le i \le n$, $A[\Psi(i)] = A[i] + 1$. The exception is $A[1] = n$, in which case the requirement is that $A[\Psi(1)] = 1$ so that Ψ is a permutation.

Grossi and Vitter use a hierarchical decomposition of A based on Ψ[2]. Let us focus on the first level of that hierarchical decomposition. Let $A_0 = A$ be the original suffix array. A bit vector $B^0[1, n]$ is defined so that $B^0[i] = 1$ iff $A[i]$ is even. Let also $\Psi_0[1, \lceil n/2 \rceil]$ contain the sequence of values $\Psi(i)$ for arguments i where $B^0[i] = 0$. Finally, let $A_1[1, \lfloor n/2 \rfloor]$ be the subsequence of $A_0[1, n]$ formed by the even $A_0[i]$ values, divided by 2.

[1] H_k is the minimum average number of bits needed to code one symbol using any compressor that fixes the code word based on the k-symbol context following the the symbol to be coded. See [6] for more formal definition.

[2] The description below follows closely the one given in [9]

Then, $A = A_0$ can be represented using only Ψ_0, B^0, and A_1. To retrieve $A[i]$, first see if $B^0[i] = 1$. If it is, then $A[i]$ is (divided by 2) somewhere in A_1. The exact position depends on how many 1's are there in B^0 up to position i, denoted $rank(B^0, i)$; that is, $A[i] = 2 \cdot A_1[rank_1(B^0, i)]$. If $B^0[i] = 0$, then $A[i]$ is odd and not represented in A_1. However, $A[i] + 1 = A[\Psi(i)]$ has to be even and thus represented in A_1. Since Ψ_0 collects only the Ψ values where $B^0[i] = 0$, it holds that $A[\Psi(i)] = A[\Psi_0[rank_0(B^0, i)]]$. Once computing $A[\Psi(i)]$ (for even $\Psi(i)$), one simply obtains $A[i] = A[\Psi(i)] - 1$.

The idea can be used recursively: Instead of representing A_1, replace it with B^2, Ψ_2, and A_2. This is continued until A_h is small enough to be represented explicitly. The complexity is $O(h)$ assuming constant-time *rank*; one can attach $o(n)$ bits data structures to a bit vector of length n such that *rank*-queries can be be answered in constant time [4,7].

It is convenient to use $h = \lceil \log \log n \rceil$, so that the $n/2^h$ entries of A_h, each of which requires $O(\log n)$ bits, take overall $O(n)$ bits. All the B^ℓ arrays add up at most $2n$ bits (as their length is halved from each level to the next), and their additional *rank* structures add $o(n)$ extra bits. The only remaining problem is how to represent the Ψ_ℓ arrays. The following regularity due to lexicographic order can be exploited:

Lemma 1 *Given a text $T[1, n]$, its suffix array $A[1, n]$, and the corresponding function Ψ, it holds $\Psi(i) < \Psi(i + 1)$ whenever $T_{A[i]} = T_{A[i+1]}$.*

This piecewise increasing property of Ψ can be used to represent each level of Ψ in $\frac{1}{2} n \log \sigma$ bits [3]. Other tradeoffs are possible using different amount of levels:

Theorem 2 (Grossi and Vitter 2005 [3]) *The Compressed Suffix Array of Grossi and Vitter supports retrieving $A[i]$ in (i) $O(\log \log n)$ time using $n \log \sigma \log \log n + O(n \log \log \sigma)$ bits of space, or (ii) $O(\log^\epsilon n)$ time using $\frac{1}{\epsilon} n \log \sigma + O(n \log \log \sigma)$ bits of space, for any $0 < \epsilon < 1$.*

As a consequence, simulating the classical binary searches [5] to find the range of suffix array containing all the occurrences of a pattern $P[1, m]$ in $T[1, n]$, can then be done in $O(m \log^{1+\epsilon} n)$ time using space proportional to the text size. Reporting the *occ* occurrence positions takes $occ \times \log^\epsilon n$ time. This can be sped up when m is large enough [3].

Grossi and Vitter also show how to modify a space-efficient suffix tree [8] so as to obtain $O(m/\log_\sigma n + \log^\epsilon n)$ search time, for any constant $0 < \epsilon < 1$, using $O(n \log \sigma)$ bits of space.

Sadakane [10] shows how the above compressed suffix array can be converted into a self-index, and at the same time optimized in several ways. He does not give direct access to $A[i]$, but rather to any prefix of $T[A[i], n]$. This still suffices to use the binary search algorithm to locate the pattern occurrences.

Sadakane represents both A and T using the full function Ψ, and a few extra structures. Imagine one wishes to compare P against $T[A[i], n]$. For the binary search, one needs to extract enough characters from $T[A[i], n]$ so that its lexicographical relation to P is clear. Retrieving character $T[A[i]]$ is easy; Use a bit vector $F[1, n]$ marking the suffixes of $A[i]$ where the first character changes from that of $A[i - 1]$. After preprocessing F for *rank*-queries, computing $j = rank_1(F, i)$ tells us that $T[A[i]] = c_j$, where c_j is the j-th smallest alphabet character. Once $T[A[i]] = c_j$ is determined this way, one needs to obtain the next character, $T[A[i] + 1]$. But $T[A[i] + 1] = T[A[\Psi(i)]]$, so one can simply move to $i' = \Psi(i)$ and keep extracting characters with the same method, as long as necessary. Note that at most $|P| = m$ characters suffice to decide a comparison with P. Thus the binary search is simulated in $O(m \log n)$ time.

Up to now the space used is $n + o(n) + \sigma \log \sigma$ bits for F and Σ. Sadakane [10] gives an improved representation for Ψ using $nH_0 + O(n \log \log \sigma)$ bits, where H_0 is the zeroth order entropy of T.

Sadakane also shows how $A[i]$ can be retrieved, by plugging in the hierarchical scheme of Grossi and Vitter. He adds to the scheme the retrieval of the inverse $A^{-1}[j]$. This is used in order to retrieve arbitrary text substrings $T[p, r]$, by first applying $i = A^{-1}[p]$ and then continuing as before to retrieve $r - p + 1$ first characters of suffix $T[A[i], n]$. This capability turns the compressed suffix array into self-index:

Theorem 3 (Sadakane [10]) *The* Compressed Suffix Array *of Sadakane is a self-index occupying $\frac{1}{\epsilon} nH_0 + O(n \log \log \sigma)$ bits, and supporting retrieval of values $A[i]$ and $A^{-1}[j]$ in $O(\log^\epsilon n)$ time, counting of pattern occurrences in $O(m \log n)$ time, and displaying any substring of T of length ℓ in $O(\ell + \log^\epsilon n)$ time. Here $0 < \epsilon \leq 1$ is an arbitrary constant.*

Grossi, Gupta, and Vitter [1,2] have improved the space-requirement of compressed suffix arrays to depend on the k-th order entropy of T. The idea behind this improvement is a more careful analysis of regularities captured by the Ψ-function when combined with the indexing capabilities of their new elegant data structure, *wavelet tree*. They obtain, among other results, the following tradeoff:

Theorem 4 (Grossi, Gupta, and Vitter 2003 [2]) *The* Compressed Suffix Array *of Grossi, Gupta, and Vitter is a self-index occupying $\frac{1}{\epsilon} nH_k + o(n \log \sigma)$ bits, and supporting retrieval of values $A[i]$ and $A^{-1}[j]$ in $O(\log^{1+\epsilon} n)$ time, counting of pattern occurrences in $O(m \log \sigma + \log^{2+\epsilon} n)$ time, and displaying any substring of T of length ℓ in $O(\ell / \log_\sigma n + \log^{1+\epsilon} n)$ time. Here $0 < \epsilon \leq 1$ is an arbitrary constant, $k \leq \alpha \log_\sigma n$ for some constant $0 < \alpha < 1$.*

In the above, value k must be fixed before building the index. Later, they notice that a simple coding of Ψ-values yields the same nH_k bound without the need of fixing k beforehand [1].

Finally, compressed suffix arrays work as building blocks to solve other CFTI problems. For example, Sadakane [11] has created a fully functional compressed suffix tree by plugging in the compressed suffix array and the space-efficient suffix tree of Munro, Raman, and Rao [8]. This compressed suffix tree occupies $O(n \log \sigma)$ bits of space, simulating all suffix tree operations with at most $O(\log n)$ slowdown.

Applications

The application domains are the same as for the classical suffix arrays and trees, with the additional advantage of scaling up to significantly larger data sets.

URL to Code

See the corresponding *Compressed Text Indexing* entry for references to compressed suffix array implementations and http://www.cs.helsinki.fi/group/suds/cst for an implementation of Sadakane's compressed suffix tree.

Cross References

▶ Compressed Text Indexing
▶ Sequential Exact String Matching
▶ Text Indexing

Recommended Reading

1. Foschini, L., Grossi, R., Gupta, A., Vitter, J.S.: When indexing equals compression: Experiments with compressing suffix arrays and applications. ACM Trans. Algorithms **2**(4), 611–639 (2006)
2. Grossi, R., Gupta, A., Vitter, J.: High-order entropy-compressed text indexes. In: Proc. 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), Baltimore, 12–14 January, pp. 841–850 (2003)
3. Grossi, R., Vitter, J.: Compressed suffix arrays and suffix trees with applications to text indexing and string matching. SIAM J. Comput. **35**(2), 378–407 (2006)
4. Jacobson, G.: Space-efficient static trees and graphs. In: Proc. 30th IEEE Symposium on Foundations of Computer Science (FOCS), Research Triangle Park, 30 October – 1 November, pp. 549–554 (1989)
5. Manber, U., Myers, G.: Suffix arrays: a new method for on-line string searches. SIAM J. Comput. **22**(5), 935–948 (1993)
6. Manzini, G.: An analysis of the Burrows-Wheeler transform. J. ACM **48**(3), 407–430 (2001)
7. Munro, I.: Tables. In: Proc. 16th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS). LNCS, vol. 1180, Hyderabad, 18–20 December, pp. 37–42 (1996)
8. Munro, I., Raman, V., Rao, S.: Space efficient suffix trees. J. Algorithms **39**(2), 205–222 (2001)
9. Navarro, G., Mäkinen, V.: Compressed full-text indexes. ACM Comput. Surv. **39**(1), Article 2 (2007)
10. Sadakane, K.: New text indexing functionalities of the compressed suffix arrays. J. Algorithms **48**(2), 294–313 (2003)
11. Sadakane, K.: Compressed suffix trees with full functionality. Theor. Comput. Syst. **41**, 589–607 (2007)

Compressed Text Indexing

2005; Ferragina, Manzini

Veli Mäkinen[1], Gonzalo Navarro[2]
[1] Department of Computer Science, University of Helsinki, Helsinki, Finland
[2] Department of Computer Science, University of Chile, Santiago, Chile

Keywords and Synonyms

Space-efficient text indexing; Compressed full-text indexing; Self-indexing

Problem Definition

Given a *text string* $T = t_1 t_2 \ldots t_n$ over an alphabet Σ of size σ, the *compressed text indexing (CTI)* problem asks to *replace* T with a space-efficient data structure capable of efficiently answering basic string matching and substring queries on T. Typical queries required from such an index are the following:

- $count(P)$: count how many times a given *pattern string* $P = p_1 p_2 \ldots p_m$ occurs in T.
- $locate(P)$: return the locations where P occurs in T.
- $display(i,j)$: return $T[i,j]$.

Key Results

An elegant solution to the problem is obtained by exploiting the connection of *Burrows-Wheeler Transform (BWT)* [1] and *Suffix Array* data structure [9]. The suffix array $SA[1,n]$ of T is the permutation of text positions $(1 \ldots n)$ listing the *suffixes* $T[i,n]$ in lexicographic order. That is, $T[SA[i], n]$ is the ith smallest suffix. The BWT is formed by (1) a permutation T^{bwt} of T defined as $T^{\text{bwt}}[i] = T[SA[i] - 1]$, where $T[0] = T[n]$, and (2) the number $i^* = SA^{-1}[1]$.

A property of the BWT is that symbols having the same context (i. e., string following them in T) are consecutive in T^{bwt}. This makes it easy to compress T^{bwt} achieving space close to high-order empirical entropies [10]. On the other hand, the suffix array is a versatile text index, allowing for example $O(m \log n)$ time counting queries (using two binary searches on SA) after which one can locate the occurrences in optimal time.

Ferragina and Manzini [3] discovered a way to combine the compressibility of the BWT and the indexing properties of the suffix array. The structure is essentially a compressed representation of the BWT plus some small additional structures to make it searchable.

We first focus on retrieving arbitrary substrings from this compressed text representation, and later consider searching capabilities. To retrieve the whole text from the structure (that is, to support $display(1, n)$), it is enough to invert the BWT. For this purpose, let us consider a table $LF[1, n]$ defined such that if $T[i]$ is permuted to $T^{\text{bwt}}[j]$ and $T[i-1]$ to $T^{\text{bwt}}[j']$ then $LF[j] = j'$. It is then immediate that T can be retrieved *backwards* by printing $T^{\text{bwt}}[i^*] \cdot T^{\text{bwt}}[LF[i^*]] \cdot T^{\text{bwt}}[LF[LF[i^*]]] \ldots$ (by definition $T^{\text{bwt}}[i^*]$ corresponds to $T[n]$).

To represent array LF space-efficiently, Ferragina and Manzini noticed that each $LF[i]$ can be expressed as follows:

Lemma 1 (Ferragina and Manzini 2005 [3]) *$LF[i] = C(c) + rank_c(i)$, where $c = T^{bwt}[i]$, $C(c)$ tells how many times symbols smaller than c appear in T^{bwt} and $rank_c(i)$ tells how many times symbol c appears in $T^{bwt}[1, i]$.*

General $display(i,j)$ queries rely on a regular sampling of the text. Every text position of the form $j' \cdot s$, being s the sampling rate, is stored together with $SA^{-1}[j' \cdot s]$, the suffix array position pointing to it. To solve $display(i,j)$ we start from the smallest sampled text position $j' \cdot s > j$ and apply the BWT inversion procedure starting with $SA^{-1}[j' \cdot s]$ instead of i^*. This gives the characters in reverse order from $j' \cdot s - 1$ to i, requiring at most $j - i + s$ steps.

It also happens that the very same two-part expression of $LF[i]$ enables efficient $count(P)$ queries. The idea is that if one knows the range of the suffix array, say $SA[sp_i, ep_i]$, such that the suffixes $T[SA[sp_i], n], T[SA[sp_i + 1], n], \ldots, T[SA[ep_i], n]$ are the only ones containing $P[i, m]$ as a prefix, then one can compute the new range $SA[sp_{i-1}, ep_{i-1}]$ where the suffixes contain $P[i-1, m]$ as a prefix, as follows: $sp_{i-1} = C(P[i-1]) + rank_{P[i-1]}(sp_i - 1) + 1$ and $ep_{i-1} = C(P[i-1]) + rank_{P[i-1]}(ep_i)$. It is then enough to scan the pattern *backwards* and compute values $C()$ and $rank_c()$ $2m$ times to find out the (possibly empty) range of the suffix array where all the suffixes start with the complete P. Returning $ep_1 - sp_1 + 1$ solves the $count(P)$ query without the need of having the suffix array available at all.

For locating each such occurrence $SA[i]$, $sp_1 \le i \le ep_1$, one can compute the sequence i, $LF[i]$, $LF[LF[i]]$, …, until $LF^k[i]$ is a sampled suffix array position and thus it is explicitly stored in the sampling structure designed for $display(i,j)$ queries. Then $SA[i] = SA[LF^k[i]] + k$. As we are virtually moving sequentially on the text, we cannot do more than s steps in this process.

Now consider the space requirement. Values $C()$ can be stored trivially in a table of $\sigma \log_2 n$ bits. $T^{\text{bwt}}[i]$ can be computed in $O(\sigma)$ time by checking for which c is $rank_c(i) \ne rank_c(i-1)$. The sampling rate can be chosen as $s = \Theta(\log^{1+\epsilon} n)$ so that the samples require $o(n)$ bits. The only real challenge is to preprocess the text for $rank_c()$ queries. This has been a subject of intensive research in recent years and many solutions have been proposed. The original proposal builds several small partial sum data structures on top of the compressed BWT, and achieves the following result:

Theorem 2 (Ferragina and Manzini 2005 [3]) *The CTI problem can be solved using a so-called* FM-Index (FMI), *of size $5nH_k + o(n \log \sigma)$ bits, that supports $count(P)$ in $O(m)$ time, $locate(P)$ in $O(\sigma \log^{1+\epsilon} n)$ time per occurrence, and display(i,j) in $O(\sigma(j - i + \log^{1+\epsilon} n))$ time. Here H_k is the* kth *order* empirical entropy *of T, $\sigma = o(\log n / \log \log n)$, $k \le \log_\sigma(n/\log n) - \omega(1)$, and $\epsilon > 0$ is an arbitrary constant.*

The original FM-Index has a severe restriction on the alphabet size. This has been removed in follow-up works. Conceptually, the easiest way to achieve a more alphabet-friendly instance of the FM-index is to build a *wavelet tree* [5] on T^{bwt}. This is a binary tree on Σ such that each node v handles a subset $S(v)$ of the alphabet, which is split among its children. The root handles Σ and each leaf handles a single symbol. Each node v encodes those positions i so that $T^{\text{bwt}}[i] \in S(v)$. For those positions, node v only stores a bit vector telling which go to the left, which to the right. The node bit vectors are preprocessed for constant time $rank_1()$ queries using $o(n)$-bit data structures [6, 12]. Grossi et al. [4] show that the wavelet tree built using the encoding of [12] occupies $nH_0 + o(n \log \sigma)$ bits. It is then easy to simulate a single $rank_c()$ query by $\log_2 \sigma$ $rank_1()$ queries. With the same cost one can obtain

$T^{\mathrm{bwt}}[i]$. Some later enhancements have improved the time requirement, so as to obtain, for example, the following result:

Theorem 3 (Mäkinen and Navarro 2005 [7]) *The CTI problem can be solved using a so-called* Succinct Suffix Array (SSA), *of size $nH_0 + o(n \log \sigma)$ bits, that supports $count(P)$ in $O(m(1 + \log \sigma / \log \log n))$ time, $locate(P)$ in $O(\log^{1+\epsilon} n \log \sigma / \log \log n)$ time per occurrence, and display(i, j) in $O((j - i + \log^{1+\epsilon} n) \log \sigma / \log \log n)$ time. Here H_0 is the zero-order entropy of T, $\sigma = o(n)$, and $\epsilon > 0$ is an arbitrary constant.*

Ferragina et al. [2] developed a technique called *compression boosting* that finds an optimal partitioning of T^{bwt} such that, when one compresses each piece separately using its zero-order model, the result is proportional to the kth order entropy. This can be combined with the idea of SSA by building a wavelet tree separately for each piece and some additional structures in order to solve global $rank_c()$ queries from the individual wavelet trees:

Theorem 4 (Ferragina et al. [4]) *The CTI problem can be solved using a so-called* Alphabet-Friendly FM-Index (AF-FMI), *of size $nH_k + o(n \log \sigma)$ bits, with the same time complexities and restrictions of SSA with $k \le \alpha \log_\sigma n$, for any constant $0 < \alpha < 1$.*

A very recent analysis [8] reveals that the space of the plain SSA is bounded by the same $nH_k + o(n \log \sigma)$ bits, making the boosting approach to achieve the same result unnecessary in theory. In practice, implementations of [4, 7] are superior by far to those building directly on this simplifying idea.

Applications

Sequence analysis in Bioinformatics, search and retrieval on oriental and agglutinating languages, multimedia streams, and even structured and traditional database scenarios.

URL to Code and Data Sets

Site Pizza-Chili http://pizzachili.dcc.uchile.cl or http://pizzachili.di.unipi.it contains a collection of standardized library implementations as well as data sets and experimental comparisons.

Cross References

► Burrows–Wheeler Transform
► Compressed Suffix Array
► Sequential Exact String Matching
► Text Indexing

Recommended Reading

1. Burrows, M., Wheeler, D.: A block sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation (1994)
2. Ferragina, P., Giancarlo, R., Manzini, G., Sciortino, M.: Boosting textual compression in optimal linear time. J. ACM **52**(4), 688–713 (2005)
3. Ferragina, P. Manzini, G.: Indexing compressed texts. J. ACM **52**(4), 552–581 (2005)
4. Ferragina, P., Manzini, G., Mäkinen, V., Navarro, G.: Compressed representation of sequences and full-text indexes. ACM Trans. Algorithms **3**(2) Article 20 (2007)
5. Grossi, R., Gupta, A., Vitter, J.: High-order entropy-compressed text indexes. In: Proc. 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 841–850 (2003)
6. Jacobson, G.: Space-efficient static trees and graphs. In: Proc. 30th IEEE Symposium on Foundations of Computer Science (FOCS), pp. 549–554 (1989)
7. Mäkinen, V., Navarro, G.: Succinct suffix arrays based on run-length encoding. Nord. J. Comput. **12**(1), 40–66 (2005)
8. Mäkinen, V., Navarro, G.: Dynamic entropy-compressed sequences and full-text indexes. In: Proc. 17th Annual Symposium on Combinatorial Pattern Matching (CPM). LNCS, vol. 4009, pp. 307–318 (2006) Extended version as TR/DCC-2006-10, Department of Computer Science, University of Chile, July 2006
9. Manber, U., Myers, G.: Suffix arrays: a new method for on-line string searches. SIAM J. Comput. **22**(5), 935–948 (1993)
10. Manzini, G.: An analysis of the Burrows-Wheeler transform. J. ACM **48**(3), 407–430 (2001)
11. Navarro, G., Mäkinen, V.: Compressed full-text indexes. ACM Comput. Surv. **39**(1) Article 2 (2007)
12. Raman, R., Raman, V., Rao, S.: Succinct indexable dictionaries with applications to encoding k-ary trees and multisets. In: Proc. 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 233–242 (2002)

Compressing Integer Sequences and Sets

2000; Moffat, Stuiver

ALISTAIR MOFFAT
Department of Computer Science and Software Engineering, University of Melbourne,
Melbourne, VIC, Australia

Problem Definition

Suppose that a message $M = \langle s_1, s_2, \ldots, s_n \rangle$ of length $n = |M|$ symbols is to be represented, where each symbol s_i is an integer in the range $1 \le s_i \le U$, for some upper limit U that may or may not be known, and may or may not be finite. Messages in this form are commonly the output of some kind of modeling step in a data compression system. The objective is to represent the message over a binary output alphabet $\{0, 1\}$ using as few as possible output bits. A special case of the problem arises when the elements of the message are strictly increasing, $s_i < s_{i+1}$. In this case the message M can be thought of as identifying a subset of $\{1, 2, \ldots, U\}$. Examples include storing sets of IP addresses or product codes, and recording the destinations of hyperlinks in the graph representation of the world wide web.

A key restriction in this problem is that it may not be assumed that $n \gg U$. That is, it must be assumed that M is too short (relative to the universe U) to warrant the calculation of an M-specific code. Indeed, in the strictly increasing case, $n \leq U$ is guaranteed. A message used as an example below is $M_1 = \langle 1, 3, 1, 1, 1, 10, 8, 2, 1, 1 \rangle$. Note that any message M can be converted to another message M' over the alphabet $U' = Un$ by taking prefix sums. The transformation is reversible, with the inverse operation known as "taking gaps".

Key Results

A key limit on static codes is expressed by the Kraft–McMillan inequality (see [13]): if the codeword for a symbol x is of length ℓ_x, then $\sum_{x=1}^{U} 2^{-\ell_x} \leq 1$ is required if the code is to be left-to-right decodeable, with no codeword a prefix of any other codeword. Another key bound is the combinatorial cost of describing a set. If an n-subset of $1 \ldots U$ is chosen at random, then a total of $\log_2 \binom{U}{n} \approx n \log_2(U/n)$ bits are required to describe that subset.

Unary and Binary Codes

As a first example method, consider *Unary coding*, in which the symbol x is represented as $x - 1$ bits that are `1`, followed by a single `0`-bit. For example, the first three symbols of message M_1 would be coded by "`0-110-0`", where the dashes are purely illustrative and do not form part of the coded representation. Because the Unary code for x is exactly x bits long, this code strongly favors small integers, and has a corresponding ideal symbol probability distribution (the distribution for which this particular pattern of codeword lengths yields the minimal message length) given by $Prob(x) = 2^{-x}$.

Unary has the useful attribute of being an infinite code. But unless the message M is dominated by small integers, Unary is a relatively expensive code. In particular, the Unary-coded representation of a message $M = \langle s_1 \ldots s_n \rangle$ requires $\sum_i s_i$ bits, and when M is a gapped representation of a subset of $1 \ldots U$, can be as long as U bits in total.

The best-known code in computing is *Binary*. If $2^{k-1} < U \leq 2^k$ for some integer k, then symbols $1 \leq s_i \leq U$ can be represented in $k \geq \log_2 U$ bits each. In this case, the code is *finite*, and the ideal probability distribution is given by $Prob(x) = 2^{-k}$. When $U = 2^k$, this then implies that $Prob(x) = 2^{-\log_2 n} = 1/n$.

When U is known precisely, and is not a power of two, $2^k - U$ of the codewords can be shortened to $k - 1$ bits long, in a *Minimal Binary* code. It is conventional to assign the short codewords to symbols $1 \cdots 2^k - U$. The codewords for the remaining symbols, $(2^k - U + 1) \cdots U$, remain k bits long.

Golomb Codes

In 1966 Solomon Golomb provided an elegant hybrid between Unary and Binary codes (see [15]). He observed that if a random n-subset of the items $1 \cdots U$ was selected, then the gaps between consecutive members of the subset were defined by a geometric probability distribution $Prob(x) = p(1-p)^{x-1}$, where $p = n/U$ is the probability that any selected item is a member of the subset.

If b is chosen such that $(1-p)^b = 0.5$, this probability distribution suggests that the codeword for $x + b$ should be one bit longer than the codeword for x. The solution $b = \log 0.5/\log(1-p) \approx 0.69/p \approx 0.69U/n$ specifies a parameter b that defines the *Golomb code*. To then represent integer x, calculate $1 + ((x-1) \operatorname{div} b)$ as a quotient, and code that part in Unary; and calculate $1 + ((x-1) \bmod b)$ as a remainder part, and code it in Minimal Binary, against a maximum bound of b. When concatenated, the two parts form the codeword for integer x. As an example, suppose that $b = 5$ is specified. Then the five Minimal Binary codewords for the five possible binary suffix parts of the codewords are "`00`", "`01`", "`10`", "`110`", and "`111`". The number 8 is thus coded as a Unary prefix of "`10`" to indicate a quotient part of 2, followed by a Minimal Binary remainder of "`10`" representing 3, to make an overall codeword of "`10-10`".

Like Unary, the Golomb code is infinite; but by design is adjustable to different probability distributions. When $b = 2^k$ for integer k a special case of the Golomb code arises, usually called a *Rice code*.

Elias Codes

Peter Elias (again, see [15]) provided further hybrids between Unary and Binary codes in work published in 1975. This family of codes are defined recursively, with Unary being the simplest member.

To move from one member of the family to the next, the previous member is used to specify the number of bits in the standard binary representation of the value x being coded (that is, the value $1 + \lfloor \log_2 x \rfloor$); then, once the length has been specified, the trailing bits of x, with the top bit suppressed, are coded in Binary.

For example, the second member of the Elias family is C_γ, and can be thought of as a Unary-Binary code: Unary to indicate the prefix part, being the magnitude of x; and then Binary to indicate the value of x within the range specified by the prefix part. The first few C_γ codewords are thus "`0`", "`10-0`", "`10-1`", "`110-00`", and so on, where the dashes are again purely illustrative. In general, the C_γ codeword for a value x requires $1 + \lfloor \log_2 x \rfloor$ bits for the Unary prefix part, and a further $\lfloor \log_2 x \rfloor$ for the binary suffix part, and the ideal probability distribution is thus given by $Prob(x) \geq 1/(2x^2)$.

After C_γ, the next member of the Elias family is C_δ. The only difference between C_γ codewords and the corresponding C_δ codewords is that in the latter C_γ is used to store the prefix part, rather than Unary. Further members of the family of Elias codes can be generated by applying the same process recursively, but for practical purposes C_δ is the last useful member of the family, even for relatively large values of x. To see why, note that $|C_\gamma(x)| \leq |C_\delta(x)|$ whenever $x \leq 31$, meaning that C_δ is longer than the next Elias code only for values $x \geq 2^{32}$.

Fibonacci-Based Codes

Another interesting code is derived from the Fibonacci sequence described (for this purpose) as $F_1 = 1$, $F_2 = 2$, $F_3 = 3$, $F_4 = 5$, $F_5 = 8$, and so on. The *Zeckendorf* representation of a natural number is a list of Fibonacci values that add up to that number, with the restriction that no two adjacent Fibonacci numbers may be used. For example, the number 10 is the sum of $2 + 8 = F_2 + F_5$.

The simplest *Fibonacci* code is derived directly from the ordered Zeckendorf representation of the target value, and consists of a "`0`" bit in the ith position (counting from the left) of the codeword if F_i does not appear in the sum, and a "`1`" bit in that position if it does, with indices considered in increasing order. Because it is not possible for both F_i and F_{i+1} to be part of the sum, the last two bits of this string must be "`01`". An appended "`1`" bit is thus sufficient to signal the end of each codeword. As always, the assumption of monotonically decreasing symbol probabilities means that short codes are assigned to small values. The code for integer one is "`1-1`", and the next few codewords are "`01-1`", "`001-1`", "`101-1`", "`0001-1`", "`1001-1`", where, as before, the embedded dash is purely illustrative.

Because $F_n \approx \phi^n$ where φ is the golden ratio $\phi = (1 + \sqrt{5})/2 \approx 1.61803$, the codeword for x is approximately $1 + \log_\phi x \approx 1 + 1.44 \log_2 x$ bits long, and is shorter than C_γ for all values except $x = 1$. It is also as good as, or better than, C_δ over a wide range of practical values between 2 and $F_{19} = 6{,}765$. Higher-order Fibonacci codes are also possible, with increased minimum codeword lengths, and decreased coefficients on the logarithmic term. Fenwick [8] provides good coverage of Fibonacci codes.

Byte Aligned Codes

Performing the necessary bit-packing and bit-unpacking operations to extract unrestricted bit sequences can be costly in terms of decoding throughput rates, and a whole class of codes that operate on units of bytes rather then bits have been developed – the *Byte Aligned* codes.

The simplest Byte Aligned code is an interleaved eight-bit analog of the Elias C_γ mechanism. The top bit in each byte is reserved for a flag that indicates (when "`0`") that "this is the last byte of this codeword" and (when "`1`") that "this is not the last byte of this codeword, take another one as well". The other seven bits in each byte are used for data bits. For example, the number 1,234 is coded into two bytes, "`209-008`", and is reconstructed via the calculation $(\texttt{209} - 128 + 1) \times 128^0 + (\texttt{008} + 1) \times 128^1 = 1,234$.

In this simplest byte aligned code, a total of $8\lceil(\log_2 x)/7\rceil$ bits are used, which makes it more effective asymptotically than the $1 + 2\lfloor \log_2 x \rfloor$ bits required by the Elias C_γ code. However, the minimum codeword length of eight bits means that Byte Aligned codes are expensive on messages dominated by small values.

Byte Aligned codes are fast to decode. They also provide another useful feature – the facility to quickly "seek" forwards in the compressed stream over a given number of codewords. A third key advantage of byte codes is that if the compressed message is to be searched, the search pattern can be rendered into a sequence of bytes using the same code, and then any byte-based pattern matching utility be invoked [7]. The zero top bit in all final bytes means that false matches are identified with a single additional test.

An improvement to the simple Byte Aligned coding mechanism arises from the observation that there is nothing special about the value 128 as the separating value between the *stopper* and *continuer* bytes, and that different values lead to different tradeoffs in overall codeword lengths [3]. In these (S, C)-Byte Aligned codes, values of S and C such that $S + C = 256$ are chosen, and each codeword consists of a sequence of zero or more continuer bytes with values greater than or equal to S, and ends with a final stopper byte with a value less than S. Other variants include methods that use bytes as the coding units to form Huffman codes, either using eight-bit coding symbols or tagged seven-bit units [7]; and methods that partially permute the alphabet, but avoid the need for a complete mapping [6]. Culpepper and Moffat [6] also describe a byte aligned coding method that creates a set of byte-based codewords with the property that the first byte uniquely identifies the length of the codeword. Similarly, *Nibble* codes can be designed as a 4-bit analog of the Byte Aligned approach, where one bit is reserved for a stopper-continuer flag, and three bits are used for data.

Other Static Codes

There have been a wide range of other variants described in the literature. Several of these adjust the code by altering the boundaries of the set of *buckets* that define the code, and coding a value x as a Unary bucket identifier, followed by a Minimal Binary offset within the specified bucket (see [15]).

For example, the Elias C_γ code can be regarded as being a Unary-Binary combination relative to a vector of bucket sizes $\langle 2^0, 2^1, 2^2, 2^3, 2^4, \ldots \rangle$. Teuhola (see [15]) proposed a hybrid in which a parameter k is chosen, and the vector of bucket sizes is given by $\langle 2^k, 2^{k+1}, 2^{k+2}, 2^{k+3}, \ldots \rangle$. One way of setting the parameter k is to take it to be the length in bits of the median sequence value, so that the first bit of each codeword approximately halves the range of observed symbol values. Another variant method is described by Boldi and Vigna [2], who use a vector $\langle 2^k - 1, (2^k - 1)2^k, (2^k - 1)2^{2k}, (2^k - 1)2^{3k}, \ldots \rangle$ to obtain a family of codes that are analytically and empirically well-suited to power-law probability distributions, especially those associated with web-graph compression. In this method k is typically in the range 2 to 4, and a Minimal Binary code is used for the suffix part.

Fenwick [8] provides detailed coverage of a wide range of static coding methods. Chen et al. [4] have also recently considered the problem of coding messages over sparse alphabets.

A Context Sensitive Code

The static codes described in the previous sections use the same set of codeword assignments throughout the encoding of the message. Better compression can be achieved in situations in which the symbol probability distribution is locally homogeneous, but not globally homogeneous.

Moffat and Stuiver [12] provided an off-line method that processes the message holisticly, in this case not because a parameter is computed (as is the case for the Binary code), but because the symbols are coded in a non-sequential manner. Their *Interpolative* code is a recursive coding method that is capable of achieving very compact representations, especially when the gaps are not independent of each other.

To explain the method, consider the subset form of the example message, as shown by sequence M_2 in Table 1. Suppose that the decoder is aware that the largest value in the subset does not exceed 29. Then every item in M is greater than or equal to $lo = 1$ and less than or equal to $hi = 29$, and the 29 different possibilities could be coded using Binary in fewer than $\lceil \log_2(29 - 1 + 1) \rceil = 5$ bits each. In particular, the mid-value in M_2, in this example the value $s_5 = 7$ (it doesn't matter which mid-value is chosen), can certainly be transmitted to the decoder using five bits. Then, once the middle number is pinned down, all of the remaining values can be coded within more precise ranges, and might require fewer than five bits each.

Now consider in more detail the range of values that the mid-value can span. Since there are $n = 10$ numbers in the list overall, there are four distinct values that precede s_5, and another five that follow it. From this argument a more restricted range for s_5 can be inferred: $lo' = lo + 4$ and $hi' = hi - 5$, meaning that the fifth value of M_2 (the number 7) can be Minimal Binary coded as a value within the range $[5, 24]$ using just 4 bits. The first row of Table 1 shows this process.

Now there are two recursive subproblems – transmitting the left part, $\langle 1, 4, 5, 6 \rangle$, against the knowledge that every value is greater than $lo = 1$ and $hi = 7 - 1 = 6$; and transmitting the right part, $\langle 17, 25, 27, 28, 29 \rangle$, against the knowledge that every value is greater than $lo = 7 + 1 = 8$ and less than or equal to $hi = 29$. These two sublists are processed recursively in the order shown in the remainder of Table 1, again with tighter ranges $[lo', hi']$ calculated and Minimal Binary codes emitted

One key aspect of the Interpolative code is that the situation can arise in which codewords that are zero bits long are called for, indicated when $lo' = hi'$. No bits need to be emitted in this case, since only one value is within the indicated range and the decoder can infer it. Four of the symbols in M_2 benefit from this possibility. This feature means that the Interpolative code is particularly effective when the subset contains clusters of consecutive items, or localized subset regions where there is a high density. In the limit, if the subset contains every element in the universal set, no bits at all are required once U is known. More generally, it is possible for dense sets to be represented in fewer than one bit per symbol.

Table 1 presents the Interpolative code using (in the final column) Minimal Binary for each value within its bounded range. A refinement is to use a Centered Minimal Binary code so that the short codewords are assigned

Compressing Integer Sequences and Sets, Table 1
Example encodings of message $M_2 = \langle 1, 4, 5, 6, 7, 17, 25, 27, 28, 29 \rangle$ using the Interpolative code. When a Minimal Binary code is used, a total of 20 bits are required. When $lo' = hi'$, no bits are output

Index i	Value s_i	lo	hi	lo'	hi'	$\{s_i - lo', hi' - lo'\}$	Binary	MinBin
5	7	1	29	5	24	2,19	`00010`	`0010`
2	4	1	6	2	4	2,2	`10`	`11`
1	1	1	3	1	3	0,2	`00`	`0`
3	5	5	6	5	5	0,0	–	–
4	6	6	6	6	6	0,0	–	–
8	27	8	29	10	27	17,17	`01111`	`11111`
6	17	8	26	8	25	9,17	`01001`	`1001`
7	25	18	26	18	26	7,8	`0111`	`1110`
9	28	28	29	28	28	0,0	–	–
10	29	29	29	29	29	0,0	–	–

in the middle of the range rather than at the beginning, recognizing that the mid value in a set is more likely to be near the middle of the range spanned by those items than it is to the ends of the range. Adding this enhancement requires a trivial restructure of Minimal Binary coding, and tends to be beneficial in practice. But improvement is not guaranteed, and, as it turns out, on sequence M_2 the use of a Centered Minimal Binary code adds one bit to the length of the compressed representation compared to the Minimal Binary code shown in Table 1.

Cheng et al. [5] describe in detail techniques for fast decoding of Interpolative codes.

Hybrid Methods

It was noted above that the message must be assumed to be short relative to the total possible universe of symbols, and that $n \ll U$. Fraenkel and Klein [9] observed that the sequence of symbol *magnitudes* (that is, the sequence of values $\lceil \log_2 s_i \rceil$) in the message must be over a much more compact and dense range than the message itself, and it can be effective to use a principled code for the prefix parts that indicate the magnitudes, in conjunction with straightforward Binary codes for the suffix parts. That is, rather than using Unary for the prefix part, a Huffman (minimum-redundancy) code can be used.

In 1996 Peter Fenwick (see [13]) described a similar mechanism using Arithmetic coding, and as well incorporated an additional benefit. His *Structured Arithmetic* coder makes use of adaptive probability estimation and two-part codes, being a magnitude and a suffix part, with both calculated adaptively. The magnitude parts have a small range, and that code is allowed to adapt its inferred probability distribution quickly, to account for volatile local probability changes. The resultant two-stage coding process has the unique benefit of "smearing" probability changes across ranges of values, rather than confining them to the actual values recently processed.

Other Coding Methods

Other recent context sensitive codes include the *Binary Adaptive Sequential* code of Moffat and Anh [11]; and the *Packed Binary* codes of Anh and Moffat [1]. More generally, Witten et al. [15] and Moffat and Turpin [13] provide details of the Huffman and Arithmetic coding techniques that are likely to yield better compression when the length of the message M is large relative to the size of the source alphabet U.

Applications

A key application of compressed set representation techniques is to the storage of inverted indexes in large full-text retrieval systems of the kind operated by web search companies [15].

Open Problems

There has been recent work on compressed set representations that support operations such as *rank* and *select*, without requiring that the set be decompressed (see, for example, Gupta et al. [10] and Raman et al. [14]). Improvements to these methods, and balancing the requirements of effective compression versus efficient data access, are active areas of research.

Experimental Results

Comparisons based on typical data sets of a realistic size, reporting both compression effectiveness and decoding efficiency are the norm in this area of work. Witten et al.[15]

give details of actual compression performance, as do the majority of published papers.

URL to Code

The page at http://www.csse.unimelb.edu.au/~alistair/codes/ provides a simple text-based "compression" system that allows exploration of the various codes described here.

Cross References

► Arithmetic Coding for Data Compression
► Compressed Text Indexing
► Rank and Select Operations on Binary Strings

Recommended Reading

1. Anh, V.N., Moffat, A.: Improved word-aligned binary compression for text indexing. IEEE Trans. Knowl. Data Eng. **18**(6), 857–861 (2006)
2. Boldi, P., Vigna, S.: Codes for the world-wide web. Internet Math. **2**(4), 405–427 (2005)
3. Brisaboa, N.R., Fariña, A., Navarro, G., Esteller, M.F.: (*S*, *C*)-dense coding: An optimized compression code for natural language text databases. In: Nascimento, M.A. (ed.) Proc. Symp. String Processing and Information Retrieval. LNCS, vol. 2857, pp. 122–136, Manaus, Brazil, October 2003
4. Chen, D., Chiang, Y.J., Memon, N., Wu, X.: Optimal alphabet partitioning for semi-adaptive coding of sources of unknown sparse distributions. In: Storer, J.A., Cohn, M. (eds.) Proc. 2003 IEEE Data Compression Conference, pp. 372–381, IEEE Computer Society Press, Los Alamitos, California, March 2003
5. Cheng, C.S., Shann, J.J.J., Chung, C.P.: Unique-order interpolative coding for fast querying and space-efficient indexing in information retrieval systems. Inf. Process. Manag. **42**(2), 407–428 (2006)
6. Culpepper, J.S., Moffat, A.: Enhanced byte codes with restricted prefix properties. In: Consens, M.P., Navarro, G. (eds.) Proc. Symp. String Processing and Information Retrieval. LNCS Volume 3772, pp. 1–12, Buenos Aires, November 2005
7. de Moura, E.S., Navarro, G., Ziviani, N., Baeza-Yates, R.: Fast and flexible word searching on compressed text. ACM Trans. Inf. Syst. **18**(2), 113–139 (2000)
8. Fenwick, P.: Universal codes. In: Sayood, K. (ed.) Lossless Compression Handbook, pp. 55–78, Academic Press, Boston (2003)
9. Fraenkel, A.S., Klein, S.T.: Novel compression of sparse bitstrings –Preliminary report. In: Apostolico, A., Galil, Z. (eds) Combinatorial Algorithms on Words, NATO ASI Series F, vol. 12, pp. 169–183. Springer, Berlin (1985)
10. Gupta, A., Hon, W.K., Shah, R., Vitter, J.S.: Compressed data structures: Dictionaries and data-aware measures. In: Storer, J.A., Cohn, M. (eds) Proc. 16th IEEE Data Compression Conference, pp. 213–222, IEEE, Snowbird, Utah, March 2006 Computer Society, Los Alamitos, CA
11. Moffat, A., Anh, V.N.: Binary codes for locally homogeneous sequences. Inf. Process. Lett. **99**(5), 75–80 (2006) Source code available from www.cs.mu.oz.au/~alistair/rbuc/
12. Moffat, A., Stuiver, L.: Binary interpolative coding for effective index compression. Inf. Retr. **3**(1), 25–47 (2000)
13. Moffat, A., Turpin, A.: Compression and Coding Algorithms. Kluwer Academic Publishers, Boston (2002)
14. Raman, R., Raman, V., Srinivasa Rao, S.: Succinct indexable dictionaries with applications to encoding *k*-ary trees and multisets. In: Proc. 13th ACM-SIAM Symposium on Discrete Algorithms, pp. 233–242, San Francisco, CA, January 2002, SIAM, Philadelphia, PA
15. Witten, I.H., Moffat, A., Bell, T.C.: Managing Gigabytes: Compressing and Indexing Documents and Images, 2nd edn. Morgan Kaufmann, San Francisco, (1999)

Compression

► Compressed Suffix Array
► Compressed Text Indexing
► Rank and Select Operations on Binary Strings
► Similarity between Compressed Strings
► Table Compression

Computational Learning

► Learning Automata

Computing Pure Equilibria in the Game of Parallel Links

2002; Fotakis, Kontogiannis, Koutsoupias, Mavronicolas, Spirakis
2003; Even-Dar, Kesselman, Mansour
2003; Feldman, Gairing, Lücking, Monien, Rode

SPYROS KONTOGIANNIS
Department of Computer Science, University of Ioannina, Ioannina, Greece

Keywords and Synonyms

Load balancing game; Incentive compatible algorithms; Nashification; Convergence of Nash dynamics

Problem Definition

This problem concerns the construction of pure Nash equilibria (PNE) in a special class of atomic congestion games, known as the Parallel Links Game (PLG). The purpose of this note is to gather recent advances in the *existence and tractability of PNE in PLG*.

THE PURE PARALLEL LINKS GAME. Let $N \equiv [n]^1$ be a set of (selfish) players, each of them willing to have her

$^1 \forall k \in \mathbb{N},\ [k] \equiv \{1, 2, \ldots, k\}$.

good served by a *unique* shared resource (link) of a system. Let $E = [m]$ be the set of all these links. For each link $e \in E$, and each player $i \in N$,let $D_{i,e}(\cdot) : \mathbb{R}_{\geq 0} \mapsto \mathbb{R}_{\geq 0}$ be the **charging mechanism** according to which link e charges player i for using it. Each player $i \in [n]$ comes with a service requirement (e. g. , traffic demand, or processing time) $W[i, e] > 0$, if she is to be served by link $e \in E$. A service requirement $W[i, e]$ is allowed to get the value ∞, to denote the fact that player i would never want to be assigned to link e. The charging mechanisms are functions of each link's cumulative congestion.

Any element $\sigma \in E$ is called a **pure strategy** for a player. Then, this player is assumed to assign her own good to link e. A collection of pure strategies for all the players is called a **pure strategies profile**, or a **configuration** of the players, or a **state** of the game.

The **individual cost** of player i wrt the profile σ is: $\mathtt{IC}_i(\sigma) = D_{i,\sigma_i}(\sum_{j\in[n]:\sigma_j=\sigma_i} W[j, \sigma_j])$. Thus, the **Pure Parallel Links Game** (PLG) is the game in strategic form defined as $\Gamma = \langle N, (\Sigma_i = E)_{i\in N}, (\mathtt{IC}_i)_{i\in N}\rangle$, whose acceptable solutions are only PNE. Clearly, an arbitrary instance of PLG can be described by the tuple $\langle N, E, (W[i, e])_{i\in N, e\in E}, (D_{i,e}(\cdot))_{i\in N, e\in E}\rangle$.

DEALING WITH SELFISH BEHAVIOR. The dominant solution concept for finite games in strategic form, is the Nash Equlibrium [14]. The definition of pure Nash Equilibria for PLG is the following:

Definition 1 (Pure Nash Equilibrium) For any instance $\langle N, E, (W[i, e])_{i\in N, e\in E}, (D_{i,e}(\cdot))_{i\in N, e\in E}\rangle$ of PLG, a pure strategies profile $\sigma \in E^n$ is a **Pure Nash Equilibrium** (PNE in short), iff:

$$\forall i \in N, \forall e \in E, \mathtt{IC}_i(\sigma) = D_{i,\sigma_i}\left(\sum\nolimits_{j\in[n]:\sigma_j=\sigma_i} W[j, \sigma_i]\right) \leq D_{i,e}\left(W[i, e] + \sum\nolimits_{j\in[n]\setminus\{i\}:\sigma_j=e} W[j, e]\right).$$

A refinement of PNE are the ***k*-robust PNE**, for $n \geq k \geq 1$ [9]. These are pure profiles for which no subset of at most k players may concurrently change their strategies in such a way that the worst possible individual cost among the movers is *strictly decreased*.

QUALITY OF PURE EQUILIBRIA. In order to determine the quality of a PNE, a social cost function that measures it must be specified. The typical assumption in the literature of PLG, is that the social cost function is the worst individual cost paid by the players: $\forall \sigma \in E^n$, $\mathtt{SC}(\sigma) = \max_{i\in N}\{\mathtt{IC}_i(\sigma)\}$ and $\forall \mathbf{p} \in (\Delta_m)^n$, $\mathtt{SC}(\mathbf{p}) = \sum_{\sigma\in E^n}(\prod_{i\in N} p_i(\sigma_i))\cdot\max_{i\in N}\{\mathtt{IC}_i(\sigma)\}$. Observe that, for mixed profiles, the social cost is the *expectation* of the maximum individual cost among the players.

The measure of the quality of an instance of PLG wrt PNE, is measured by the **Pure Price of Anarchy** ($\mathtt{PPoA}$ in short) [12]: $\mathtt{PPoA} = \max\{(\mathtt{SC}(\sigma))/\mathtt{OPT} : \sigma \in E^n \text{ is PNE}\}$ where $\mathtt{OPT} \equiv \min_{\sigma\in E^n}\{\mathtt{SC}(\sigma)\}$.

DISCRETE DYNAMICS. Crucial concepts of strategic games are the best and better responses. Given a configuration $\sigma \in E^n$, an **improvement step**, or **selfish step**, or **better response** of player $i \in N$ is the choice by i of a pure strategy $\alpha \in E \setminus \{\sigma_i\}$, so that player i would have a positive gain by this *unilateral* change (i. e., provided that the other players maintain the same strategies). That is, $\mathtt{IC}_i(\sigma) > \mathtt{IC}_i(\sigma \oplus_i \alpha)$ where, $\sigma \oplus_i \alpha \equiv (\sigma_1, \ldots, \sigma_{i-1}, \alpha, \sigma_{i+1}, \ldots, \sigma_n)$. A **best response**, or **greedy selfish step** of player i, is any change from the current link σ_i to an element $\alpha^* \in \arg\max_{a\in E}\{\mathtt{IC}_i(\sigma \oplus_i \alpha)\}$. An **improvement path** (aka a **sequence of selfish steps** [6], or an **elementary step system** [3]) is a sequence of configurations $\pi = \langle\sigma(1), \ldots, \sigma(k)\rangle$ such that

$$\forall 2 \leq r \leq k, \exists i_r \in N, \exists \alpha_r \in E: \\ [\sigma(r) = \sigma(r-1)\oplus_{i_r}\alpha_r]\wedge[\mathtt{IC}_{i_r}(\sigma(r)) < \mathtt{IC}_{i_r}(\sigma(r-1))].$$

A game has the **Finite Improvement Property** (FIP) iff any improvement path has *finite* length. A game has the **Finite Best Response Property** (FBRP) iff any improvement path, each step of whose is a best response of some player, has *finite* length.

An alternative trend is to, rather than consider *sequential* improvement paths, let the players conduct selfish improvement steps *concurrently*. Nevertheless, the selfish decisions are no longer deterministic, but rather distributions over the links, in order to have some notion of a priori Nash property that justifies these moves. The selfish players try to minimize their *expected* individual costs this time. Rounds of concurrent moves occur until a posteriori Nash Property is achieved. This is called a **selfish rerouting** policy [4].

Subclasses of PLG

[PLG$_1$] Monotone PLG: The charging mechanism of each pair of a link and a player, is a *non–decreasing function* of the resource's cumulative congestion.

[PLG$_2$] Resource Specific Weights PLG (RSPLG): Each player may have a different service demand from every link.

[PLG$_3$] Player Specific Delays PLG (PSPLG): Each link may have a different charging mechanism for each player. Some special cases of PSPLG are the following:

[PLG$_{3.1}$] Linear Delays PSPLG: Every link has a (player specific) *affine* charging mechanism: $\forall i \in N$, $\forall e \in E, D_{i,e}(x) = a_{i,e}x + b_{i,e}$ for some $a_{i,e} > 0$ and $b_{i,e} \geq 0$.

[PLG$_{3.1.1}$] Related Delays PSPLG: Every link has a (player specific) *non–uniformly related* charging mechanism: $\forall i \in N, \forall e \in E, W[i,e] = w_i$ and $D_{i,e}(x) = a_{i,e}x$ for some $a_{i,e} > 0$.

[PLG$_4$] Resource Uniform Weights PLG (RUPLG): Each player has a unique service demand from all the resources. Ie, $\forall i \in N, \forall e \in E, W[i,e] = w_e > 0$. A special case of RUPLG is:

[PLG$_{4.1}$] Unweighted PLG: All the players have identical demands from all the links: $\forall i \in N, \forall e \in E, W[i,e] = 1$.

[PLG$_5$] Player Uniform Delays PLG (PUPLG): Each resource adopts a unique charging mechanism, for all the players. That is, $\forall i \in N, \forall e \in E, D_{i,e}(x) = d_e(x)$.

[PLG$_{5.1}$] Unrelated Parallel Machines, or **Load Balancing PLG (LBPLG)**: The links behave as parallel machines. That is, they charge each of the players for the cumulative load assigned to their hosts. One may think (wlog) that all the machines have as charging mechanisms the identity function. That is, $\forall i \in N, \forall e \in E, D_{i,e}(x) = x$.

[PLG$_{5.1.1}$] Uniformly Related Machines LBPLG: Each player has the same demand at every link, and each link serves players at a fixed rate. That is: $\forall i \in N, \forall e \in E, W[i,e] = w_i$ and $D_{i,e}(x) = \frac{x}{s_e}$. Equivalently, service demands proportional to the capacities of the machines are allowed, but the identity function is required as the charging mechanism: $\forall i \in N, \forall e \in E, W[i,e] = \frac{w_i}{s_e}$ and $D_{i,e}(x) = x$.

[PLG$_{5.1.1.1}$] Identical Machines LBPLG: Each player has the same demand at every link, and all the delay mechanisms are the identity function: $\forall i \in N, \forall e \in E, W[i,e] = w_i$ and $D_{i,e}(x) = x$.

[PLG$_{5.1.2}$] Restricted Assignment LBPLG: Each traffic demand is either of unit or infinite size. The machines are identical. Ie, $\forall i \in N, \forall e \in E, W[i,e] \in \{1, \infty\}$ and $D_{i,e}(x) = x$.

Algorithmic Questions concerning PLG

The following algorithmic questions are considered:

Problem 1 (PNEExistsInPLG(E, N, W, D))
INPUT:
An instance $\langle N, E, (W[i,e])_{i\in N, e\in E}, (D_{i,e}(\cdot))_{i\in N, e\in E}\rangle$ *of PLG*
OUTPUT: *Is there a configuration* $\sigma \in E^n$ *of the players to the links, which is a PNE?*

Problem 2 (PNEConstructionInPLG(E, N, W, D))
INPUT:
An instance $\langle N, E, (W[i,e])_{i\in N, e\in E}, (D_{i,e}(\cdot))_{i\in N, e\in E}\rangle$ *of PLG*
OUTPUT: *An assignment* $\sigma \in E^n$ *of the players to the links, which is a PNE.*

Problem 3 (BestPNEInPLG(E, N, W, D))
INPUT:
An instance $\langle N, E, (W[i,e])_{i\in N, e\in E}, (D_{i,e}(\cdot))_{i\in N, e\in E}\rangle$ *of PLG. A* ***social cost*** *function* $SC : (\mathbb{R}_{\geq 0})^m \mapsto \mathbb{R}_{\geq 0}$ *that characterizes the quality of any configuration* $\sigma \in E^N$.
OUTPUT: *An assignment* $\sigma \in E^n$ *of the players to the links, which is a PNE and minimizes the value of the social cost, compared to other PNE of PLG.*

Problem 4 (WorstPNEInPLG(E, N, W, D))
INPUT:
An instance $\langle N, E, (W[i,e])_{i\in N, e\in E}, (D_{i,e}(\cdot))_{i\in N, e\in E}\rangle$ *of PLG. A* ***social cost*** *function* $SC : (\mathbb{R}_{\geq 0})^m \mapsto \mathbb{R}_{\geq 0}$ *that characterizes the quality of any configuration* $\sigma \in E^N$.
OUTPUT: *An assignment* $\sigma \in E^n$ *of the players to the links, which is a PNE and maximizes the value of the social cost, compared to other PNE of PLG.*

Problem 5 (DynamicsConvergeInPLG(E, N, W, D))
INPUT:
An instance $\langle N, E, (W[i,e])_{i\in N, e\in E}, (D_{i,e}(\cdot))_{i\in N, e\in E}\rangle$ *of PLG*
OUTPUT: *Does FIP (or FBRP) hold? How long does it take then to reach a PNE?*

Problem 6 (ReroutingConvergeInPLG(E, N, W, D))
INPUT:
An instance $\langle N, E, (W[i,e])_{i\in N, e\in E}, (D_{i,e}(\cdot))_{i\in N, e\in E}\rangle$ *of PLG*
OUTPUT: *Compute (if any) a selfish rerouting policy that converges to a PNE.*

Status of Problem 1

Player uniform, unweighted atomic congestion games always possess a PNE [15], with no assumption on monotonicity of the charging mechanisms. Thus, Problem 1 is already answered for all unweighted PUPLG. Nevertheless, this is not necessarily the case for weighted versions of PLG:

Theorem 1 ([13]) *There is an instance of (monotone) PSPLG with only three players and three strategies per player, possessing no PNE. On the other hand, any unweighted instance of monotone PSPLG possesses at least one PNE.*

Similar (positive) results were given for LBPLG. The key observation that lead to these results, is the fact that the lexicographically minimum vector of machine loads is always a PNE of the game.

Theorem 2 *There is always a PNE for any instance of Uniformly Related LBPLG* [7]*, and actually for any instance of LBPLG* [3]. Indeed, there is a $k-$robust PNE for any instance of LBPLG, and any $1 \leq k \leq n$ [9].

Status of Problems 2, 5 and 6

[13] gave a constructive proof of existence for PNE in unweighted, monotone PSPLG, and thus implies a path of length at most n that leads to a PNE. Although this is a very efficient construction of PNE, it is not necessarily an improvement path, when all players are considered to coexist all the time, and therefore there is no justification for the adoption of such a path by the players. Milchtaich [13] proved that from an arbitrary initial configuration and allowing only best reply defections, there is a best reply improvement path of length at most $m \cdot \binom{n+1}{2}$. Finally, [11] proved for unweighted, Related PSPLG that it possesses FIP. Nevertheless, the convergence time is poor.

For LBPLG, the implicit connection of PNE construction to classical scheduling problems, has lead to quite interesting results.

Theorem 3 ([7]) *The LPT algorithm of Graham, yields a PNE for the case of Uniformly Related LBPLG, in time $\mathcal{O}(m \log m)$.*

The drawback of the LPT algorithm is that it is centralized and not selfishly motivated. An alternative approach, called **Nashification**, is to start from an arbitrary initial configuration $\sigma \in E^n$ and then try to construct a PNE of at most the same maximum individual cost among the players.

Theorem 4 ([6]) *There is an $\mathcal{O}(nm^2)$ time Nashification algorithm for any instance of Uniformly Related PLG.*

An alternative style of Nashification, is to let the players follow an arbitrary improvement path. Nevertheless, it is not always the case that this leads to a polynomial time construction of a PNE, as the following theorem states:

Theorem 5 *For Identical Machines LBPLG:*

- *There exist best response improvement paths of length $\Omega\left(\max\left\{2^{\sqrt{n}}, \left(\frac{n}{m^2}\right)^m\right\}\right)$* [3,6].
- *Any best response improvement path is of length $\mathcal{O}(2^n)$* [6].
- *Any best response improvement path, which gives priority to players of maximum weight among those willing to defect in each improvement step, is of length at most n* [3].
- *If all the service demands are integers, then any improvement path which gives priority to unilateral improvement steps, and otherwise allows only selfish 2-flips (ie, swapping of hosting machines between two goods) converges to a 2-robust PNE in at most $\frac{1}{2}(\sum_{i \in N} w_i)^2$ steps* [9].

The following result concerns selfish rerouting policies:

Theorem 6 ([4])

- *For unweighted Identical Machines LBPLG, a simple policy (BALANCE) forcing all the players of overloaded links to migrate to a new (random) link with probability proportional to the load of the link, converges to a PNE in $\mathcal{O}(\log \log n + \log m)$ rounds of concurrent moves. The same convergence time holds also for a simple Nash Rerouting Policy, in which each mover actually has an incentive to move.*
- *For unweighted Uniformly Related LBPLG, BALANCE has the same convergence time, but the Nash Rerouting Policy may converge in $\Omega\left(\sqrt{n}\right)$ rounds.*

Finally, a generic result of [5] is mentioned, that computes a PNE for arbitrary *unweighted, player uniform* symmetric network congestion games in polynomial time, by a nice exploitation of Rosenthal's potential and the solution of a proper minimum cost flow problem. Therefore, for PLG the following result is implied:

Theorem 7 ([5]) *For unweighted, monotone PUPLG, a PNE can be constructed in polynomial time.*

Of course, this result provides no answer, e. g., for Restricted Assignment LBPLG, for which it is still not known how to efficiently compute PNE.

Status of Problems 3 and 4

The proposed LPT algorithm of [7] for constructing PNE in Uniformly Related LBPLG, actually provides a solution which is at most $1.52 < \text{PPoA}(LPT) < 1.67$ times worse than the optimum PNE (which is indeed the allocation of the goods to the links that minimizes the make-span). The construction of the optimum, as well as the worst PNE are hard problems, which nevertheless admits a PTAS (in some cases):

Theorem 8 *For LBPLG with a social cost function as defined in the* QUALITY OF PURE EQUILIBRIA *paragraph:*

- *For Identical Machines, constructing the optimum or the worst PNE is* **NP**$-$*hard* [7].
- *For Uniformly Related Machines, there is a PTAS for the optimum PNE* [6].

- *For Uniformly Related Machines, it holds that* $PPoA = \Theta\left(\min\{(\log m)/(\log\log m), \log(s_{\max})/(s_{\min})\}\right)$ [2].
- *For the Restricted Assignments,* $PPoA = \Omega((\log m)/(\log\log m))$ [10].
- *For a generalization of the Restricted Assignments, where the players have goods of any positive, otherwise infinite service demands from the links (and not only elements of* $\{1, \infty\}$*), it holds that* $m - 1 \leq PPoA < m$ [10].

It is finally mentioned that a recent result [1] for unweighted, single commodity network congestion games with linear delays, is translated to the following result for PLG:

Theorem 9 ([1]) *For unweighted PUPLG with linear charging mechanisms for the links, the worst case PNE may be a factor of* $PPoA = 5/2$ *away from the optimum solution, wrt the social cost defined in the* QUALITY OF PURE EQUILIBRIA *paragraph.*

Key Results

None

Applications

Congestion games in general have attracted much attention from many disciplines, partly because they capture a large class of routing and resource allocation scenarios.

PLG in particular, is the most elementary (non–trivial) atomic congestion game among a large number of players. Despite its simplicity, it was proved ([8] that it is asymptotically the worst case instance wrt the maximum individual cost measure, for a large class atomic congestion games involving the so called *layered networks*. Therefore, PLG is considered an excellent starting point for studying congestion games in large scale networks.

The importance of seeking for PNE, rather than arbitrary (mixed in general) NE, is quite obvious in sciences like the economics, ecology, and biology. It is also important for computer scientists, since it enforces deterministic costs to the players, and both the players and the network designer may feel safer in this case about what they will actually have to pay.

The question whether the Nash Dynamics converge to a PNE in a reasonable amount of time, is also quite important, since (in case of a positive answer) it justifies the selfish, decentralized, local dynamics that appear in large scale communications systems. Additionally, the selfish rerouting schemes are of great importance, since this is what should actually be expected from selfish, decentralized computing environments.

Open Problems

Open Question 1 *Determine the (in)existence of PNE for all the instances of PLG that do not belong in LBPLG, or in monotone PSPLG.*

Open Question 2 *Determine the (in)existence of* k*−robust PNE for all the instances of PLG that do not belong in LBPLG.*

Open Question 3 *Is there a polynomial time algorithm for constructing* k*−robust PNE, even for the Identical Machines LBPLG and* $k \geq 1$ *being a constant?*

Open Question 4 *Do the improvement paths of instances of PLG other than PSPLG and LBPLG converge to a PNE?*

Open Question 5 *Are there selfish rerouting policies of instances of PLG other than Identical Machines LBPLG converge to a PNE? How long much time would they need, in case of a positive answer?*

Cross References

▶ Best Response Algorithms for Selfish Routing
▶ Price of Anarchy
▶ Selfish Unsplittable Flows: Algorithms for Pure Equilibria

Recommended Reading

1. Christodoulou, G., Koutsoupias, E.: The Price of Anarchy of Finite Congestion Games. In: Proc. of the 37th ACM Symp. on Th. of Comp. (STOC '05), pp. 67–73. ACM, Baltimore (2005)
2. Czumaj, A., Vöcking, B.: Tight bounds for worst-case equilibria. In: Proc. of the 13th ACM-SIAM Symp. on Discr. Alg. (SODA '02), pp. 413–420. SIAM, San Francisco (2002)
3. Even-Dar, E., Kesselman, A., Mansour, Y.: Convergence time to nash equilibria. In: Proc. of the 30th Int. Col. on Aut., Lang. and Progr. (ICALP '03). LNCS, pp. 502–513. Springer, Eindhoven (2003)
4. Even-Dar, E., Mansour, Y.: Fast convergence of selfish rerouting. In: Proc. of the 16th ACM-SIAM Symp. on Discr. Alg. (SODA '05), SIAM, pp. 772–781. SIAM, Vancouver (2005)
5. Fabrikant, A., Papadimitriou, C., Talwar, K.: The complexity of pure nash equilibria. In: Proc. of the 36th ACM Symp. on Th. of Comp. (STOC '04). ACM, Chicago (2004)
6. Feldmann, R., Gairing, M., Lücking, T., Monien, B., Rode, M.: Nashification and the coordination ratio for a selfish routing game. In: Proc. of the 30th Int. Col. on Aut., Lang. and Progr. (ICALP '03). LNCS, pp. 514–526. Springer, Eindhoven (2003)
7. Fotakis, D., Kontogiannis, S., Koutsoupias, E., Mavronicolas, M., Spirakis, P.: The structure and complexity of nash equilibria

for a selfish routing game. In: Proc. of the 29th Int. Col. on Aut., Lang. and Progr. (ICALP '02). LNCS, pp. 123–134. Springer, Málaga (2002)

8. Fotakis, D., Kontogiannis, S., Spirakis, P.: Selfish unsplittable flows. Theor. Comput. Sci. **348**, 226–239 (2005) Special Issue dedicated to ICALP (2004) (TRACK-A)
9. Fotakis, D., Kontogiannis, S., Spirakis, P.: Atomic congestion games among coalitions. In: Proc. of the 33rd Int. Col. on Aut., Lang. and Progr. (ICALP '06). LNCS, vol. 4051, pp. 572–583. Springer, Venice (2006)
10. Gairing, M., Luecking, T., Mavronicolas, M., Monien, B.: The price of anarchy for restricted parallel links. Parallel Process. Lett. **16**, 117–131 (2006) Preliminary version appeared in STOC 2004
11. Gairing, M., Monien, B., Tiemann, K.: Routing (un-)splittable flow in games with player specific linear latency functions. In: Proc. of the 33rd Int. Col. on Aut., Lang. and Progr. (ICALP '06). LNCS, pp. 501–512. Springer, Venice (2006)
12. Koutsoupias, E., Papadimitriou, C.: Worst-case equilibria. In: Proc. of the 16th Annual Symp. on Theor. Aspects of Comp. Sci. (STACS '99), pp. 404–413. Springer, Trier (1999)
13. Milchtaich, I.: Congestion games with player-specific payoff functions. Games Econ. Behav. **13**, 111–124 (1996)
14. Nash, J.: Noncooperative games. Annals Math. **54**, 289–295 (1951)
15. Rosenthal, R.: A class of games possessing pure-strategy nash equilibria. Int. J. Game Theory **2**, 65–67 (1973)

Concurrent Programming, Mutual Exclusion

1965; Dijkstra

Gadi Taubenfeld
Department of Computer Science, Interdisciplinary Center Herzlia, Herzliya, Israel

Keywords and Synonyms

Critical section problem

Problem Definition

Concurrency, Synchronization and Resource Allocation

A *concurrent* system is a collection of processors that communicate by reading and writing from a shared memory. A distributed system is a collection of processors that communicate by sending messages over a communication network. Such systems are used for various reasons: to allow a large number of processors to solve a problem together much faster than any processor can do alone, to allow the distribution of data in several locations, to allow different processors to share resources such as data items, printers or discs, or simply to enable users to send electronic mail.

A *process* corresponds to a given computation. That is, given some program, its execution is a process. Sometimes, it is convenient to refer to the program code itself as a process. A process runs on a *processor*, which is the physical hardware. Several processes can run on the same processor although in such a case only one of them may be active at any given time. Real concurrency is achieved when several processes are running simultaneously on several processors.

Processes in a concurrent system often need to synchronize their actions. *Synchronization* between processes is classified as either cooperation or contention. A typical example for cooperation is the case in which there are two sets of processes, called the producers and the consumers, where the producers produce data items which the consumers then consume.

Contention arises when several processes compete for exclusive use of shared resources, such as data items, files, discs, printers, etc. For example, the integrity of the data may be destroyed if two processes update a common file at the same time, and as a result, deposits and withdrawals could be lost, confirmed reservations might have disappeared, etc. In such cases it is sometimes essential to allow at most one process to use a given resource at any given time.

Resource allocation is about interactions between processes that involve contention. The problem is, how to resolve conflicts resulting when several processes are trying to use shared resources. Put another way, how to allocate shared resources to competing processes. A special case of a general resource allocation problem is the *mutual exclusion* problem where only a single resource is available.

The Mutual Exclusion Problem

The *mutual exclusion* problem, which was first introduced by Edsger W. Dijkstra in 1965, is the guarantee of mutually exclusive access to a single shared resource when there are several competing processes [6]. The problem arises in operating systems, database systems, parallel supercomputers, and computer networks, where it is necessary to resolve conflicts resulting when several processes are trying to use shared resources. The problem is of great significance, since it lies at the heart of many interprocess synchronization problems.

The problem is formally defined as follows: it is assumed that each process is executing a sequence of instructions in an infinite loop. The instructions are divided into four continuous sections of code: the *remainder, entry, critical section* and *exit*. Thus, the structure of a mutual exclusion solution looks as follows:

```
loop forever
    remainder code;
    entry code;
    critical section;
    exit code
end loop
```

A process starts by executing the remainder code. At some point the process might need to execute some code in its critical section. In order to access its critical section a process has to go through an entry code which guarantees that while it is executing its critical section, no other process is allowed to execute its critical section. In addition, once a process finishes its critical section, the process executes its exit code in which it notifies other processes that it is no longer in its critical section. After executing the exit code the process returns to the remainder.

The Mutual exclusion problem is to write the code for the *entry code* and the *exit code* in such a way that the following two basic requirements are satisfied.

Mutual exclusion: *No two processes are in their critical sections at the same time.*

Deadlock-freedom: *If a process is trying to enter its critical section, then some process, not necessarily the same one, eventually enters its critical section.*
The deadlock-freedom property guarantees that the system as a whole can always continue to make progress. However deadlock-freedom may still allow "starvation" of individual processes. That is, a process that is trying to enter its critical section, may never get to enter its critical section, and wait forever in its entry code. A stronger requirement, which does not allow starvation, is defined as follows.

Starvation-freedom: *If a process is trying to enter its critical section, then this process must eventually enter its critical section.*
Although starvation-freedom is strictly stronger than deadlock-freedom, it still allows processes to execute their critical sections arbitrarily many times before some trying process can execute its critical section. Such a behavior is prevented by the following fairness requirement.

First-in-first-out (FIFO): *No beginning process can enter its critical section before a process that is already waiting for its turn to enter its critical section.*
The first two properties, mutual exclusion and deadlock freedom, were required in the original statement of the problem by Dijkstra. They are the minimal requirements that one might want to impose. In solving the problem, it is assumed that once a process starts executing its critical section the process always finishes it regardless of the activity of the other processes. Of all the problems in interprocess synchronization, the mutual exclusion problem is the one studied most extensively. This is a deceptive problem, and at first glance it seems very simple to solve.

Key Results

Numerous solutions for the problem have been proposed since it was first introduced by Edsger W. Dijkstra in 1965 [6]. Because of its importance and as a result of new hardware and software developments, new solutions to the problem are still being designed. Before the results are discussed, few models for interprocess communication are mentioned.

Atomic Operations

Most concurrent solutions to the problem assumes an architecture in which n processes communicate asynchronously via a shared objects. All architectures support *atomic registers*, which are shared objects that support atomic reads and writes operations. A weaker notion than an atomic register, called a *safe* register, is also considered in the literature. In a safe register, a read not concurrent with any writes must obtain the correct value, however, a read that is concurrent with some write, may return an arbitrary value. Most modern architectures support also some form of atomicity which is stronger than simple reads and writes. Common atomic operations have special names. Few examples are,

- *Test-and-set*: takes a shared registers r and a value *val*. The value *val* is assigned to r, and the old value of r is returned.
- *Swap*: takes a shared registers r and a local register ℓ, and atomically exchange their values.
- *Fetch-and-increment*: takes a register r. The value of r is incremented by 1, and the old value of r is returned.
- *Compare-and-swap*: takes a register r, and two values: *new* and *old*. If the current value of the register r is equal to *old*, then the value of r is set to *new* and the value *true* is returned; otherwise r is left unchanged and the value *false* is returned.

Modern operating systems (such as Unix and Windows) implement synchronization mechanisms, such as semaphores, that simplify the implementation of mutual exclusion locks and hence the design of concurrent applications. Also, modern programming languages (such as Modula and Java) implement the monitor concept which is a program module that is used to ensure exclusive access to resources.

Algorithms and Lower Bounds

There are hundreds of beautiful algorithms for solving the problem some of which are also very efficient. Only few are mentioned below. First algorithms that use only atomic registers, or even safe registers, are discussed.

The Bakery Algorithm. The Bakery algorithm is one of the most known and elegant mutual exclusion algorithms using only safe registers [9]. The algorithm satisfies the FIFO requirement, however it uses unbounded size registers. A modified version, called the Black-White Bakery algorithm, satisfies FIFO and uses bounded number of bounded size atomic registers [14].

Lower bounds. A space lower bound for solving mutual exclusion using only atomic registers is that: any deadlock-free mutual exclusion algorithm for n processes must use at least n shared registers [5]. It was also shown in [5] that this bound is tight. A time lower bound for any mutual exclusion algorithm using atomic registers is that: there is no a priori bound on the number of steps taken by a process in its entry code until it enters its critical section (counting steps only when no other process is in its critical section or exit code) [2]. Many other interesting lower bounds exist for solving mutual exclusion.

A Fast Algorithm. A *fast* mutual exclusion algorithm, is an algorithm in which in the absence of contention only a constant number of shared memory accesses to the shared registers are needed in order to enter and exit a critical section. In [10], a fast algorithm using atomic registers is described, however, in the presence of contention, the winning process may have to check the status of all other n processes before it is allowed to enter its critical section. A natural question to ask is whether this algorithm can be improved for the case where there is contention.

Adaptive Algorithms. Since the other contending processes are waiting for the winner, it is particularly important to speed their entry to the critical section, by the design of an *adaptive* mutual exclusion algorithm in which the time complexity is independent of the total number of processes and is governed only by the current degree of contention. Several (rather complex) adaptive algorithms using atomic registers are known [1,3,14]. (Notice that, the time lower bound mention earlier implies that no adaptive algorithm using only atomic registers exists when time is measured by counting *all* steps.)

Local-spinning Algorithms. Many algorithms include busy-waiting loops. The idea is that in order to wait, a process *spins* on a flag register, until some other process terminates the spin with a single write operation. Unfortunately, under contention, such spinning may generate lots of traffic on the interconnection network between the process and the memory. An algorithm satisfies local spinning if the only type of spinning required is local spinning. Local Spinning is the situation where a process is spinning on locally-accessible registers. Shared registers may be locally-accessible as a result of either coherent caching or when using distributed shared memory where shared memory is physically distributed among the processors.

Three local-spinning algorithms are presented in [4,8,11]. These algorithms use strong atomic operations (i. e., fetch-and-increment, swap, compare-and-swap), and are also called *scalable* algorithms since they are both local-spinning and adaptive. Performance studies done, have shown that these algorithms scale very well as contention increases. Local spinning algorithms using only atomic registers are presented in [1,3,14].

Only few representative results have been mentioned. There are dozens of other very interesting algorithms and lower bounds. All the results discussed above, and many more, are described details in [15]. There are also many results for solving mutual exclusion in distributed message passing systems [13].

Applications

Synchronization is a fundamental challenge in computer science. It is fast becoming a major performance and design issue for concurrent programming on modern architectures, and for the design of distributed and concurrent systems.

Concurrent access to resources shared among several processes must be synchronized in order to avoid interference between conflicting operations. Mutual exclusion *locks* (i. e., algorithms) are the de facto mechanism for concurrency control on concurrent applications: a process accesses the resource only inside a critical section code, within which the process is guaranteed exclusive access. The popularity of this approach is largely due the apparently simple programming model of such locks and the availability of implementations which are efficient and scalable. Essentially all concurrent programs (including operating systems) use various types of mutual exclusion locks for synchronization.

When using locks to protect access to a resource which is a large data structure (or a database), the *granularity* of synchronization is important. Using a single lock to protect the whole data structure, allowing only one process at a time to access it, is an example of *coarse-grained* synchronization. In contrast, *fine-grained* synchronization enables to lock "small pieces" of a data structure, allowing several processes with non-interfering operations to access it concurrently. Coarse-grained synchronization is easier to program but is less efficient and is not fault-tolerant compared to fine-grained synchronization. Using locks may degrade performance as it enforces processes to wait for a lock to be released. In few cases of simple data structures, such as queues, stacks and counters, locking may be avoided by using lock-free data structures.

Cross References

▶ Registers
▶ Self-Stabilization

Recommended Reading

In 1968, Edsger Wybe Dijkstra has published his famous paper "Co-operating sequential processes" [7], that originated the field of concurrent programming. The mutual exclusion problem was first stated and solved by Dijkstra in [6], where the first solution for two processes, due to Dekker, and the first solution for n processes, due to Dijkstra, have appeared. In [12], a collection of some early algorithms for mutual exclusion are described. In [15], dozens of algorithms for solving the mutual exclusion problems and wide variety of other synchronization problems are presented, and their performance is analyzed according to precise complexity measures.

1. Afek, Y., Stupp, G., Touitou, D.: Long lived adaptive splitter and applications. Distrib. Comput. **30**, 67–86 (2002)
2. Alur, R., Taubenfeld, G.: Results about fast mutual exclusion. In: Proceedings of the 13th IEEE Real-Time Systems Symposium, December 1992, pp. 12–21
3. Anderson, J.H., Kim, Y.-J.: Adaptive mutual exclusion with local spinning. In: Proceedings of the 14th international symposium on distributed computing. Lect. Notes Comput. Sci. **1914**, 29–43, (2000)
4. Anderson, T.E.: The performance of spin lock alternatives for shared-memory multiprocessor. IEEE Trans. Parallel Distrib. Syst. **1**(1), 6–16 (1990)
5. Burns, J.N., Lynch, N.A.: Bounds on shared-memory for mutual exclusion. Inform. Comput. **107**(2), 171–184 (1993)
6. Dijkstra, E.W.: Solution of a problem in concurrent programming control. Commun. ACM **8**(9), 569 (1965)
7. Dijkstra, E.W.: Co-operating sequential processes. In: Genuys, F. (ed.) Programming Languages, pp. 43–112. Academic Press, New York (1968). Reprinted from: Technical Report EWD-123, Technological University, Eindhoven (1965)
8. Graunke, G., Thakkar, S.: Synchronization algorithms for shared-memory multiprocessors. IEEE Comput. **28**(6), 69–69 (1990)
9. Lamport, L.: A new solution of Dijkstra's concurrent programming problem. Commun. ACM **17**(8), 453–455 (1974)
10. Lamport, L.: A fast mutual exclusion algorithm. ACM Trans. Comput. Syst. **5**(1), 1–11 (1987)
11. Mellor-Crummey, J.M., Scott, M.L.: Algorithms for scalable synchronization on shared-memory multiprocessors. ACM Trans. Comput. Syst. **9**(1), 21–65 (1991)
12. Raynal, M.: Algorithms for mutual exclusion. MIT Press, Cambridge (1986). Translation of: Algorithmique du parallélisme, (1984)
13. Singhal, M.: A taxonomy of distributed mutual exclusion. J. Parallel Distrib. Comput. **18**(1), 94–101 (1993)
14. Taubenfeld, G.: The black-white bakery algorithm. In: 18th international symposium on distributed computing, October (2004). LNCS, vol. 3274, pp. 56–70. Springer, Berlin (2004)
15. Taubenfeld, G.: Synchronization algorithms and concurrent programming. Pearson Education – Prentice-Hall, Upper Saddle River (2006) ISBN: 0131972596

Connected Dominating Set

2003; Cheng, Huang, Li, Wu, Du

XIUZHEN CHENG[1], FENG WANG[2], DING-ZHU DU[3]
[1] Department of Computer Science, The George Washington University, Washington, D.C., USA
[2] Mathematical Science and Applied Computing, Arizona State University at the West Capmus, Phoenix, AZ, USA
[3] Department of Computer Science, University of Dallas at Texas, Richardson, TX, USA

Keywords and Synonyms

Techniques for partition

Problem Definition

Consider a graph $G = (V, E)$. A subset C of V is called a *dominating set* if every vertex is either in C or adjacent to a vertex in C. If, furthermore, the subgraph induced by C is connected, then C is called a *connected dominating set*. A connected dominating set with a minimum cardinality is called a *minimum connected dominating set (MCDS)*. Computing a MCDS is an NP-hard problem and there is no polynomial-time approximation with performance ratio $\rho H(\Delta)$ for $\rho < 1$ unless $NP \subseteq DTIME(n^{O(\ln \ln n)})$ where H is the harmonic function and Δ is the maximum degree of the input graph [10].

A unit disk is a disk with radius one. A *unit disk graph (UDG)* is associated with a set of unit disks in the Euclidean plane. Each node is at the center of a unit disk. An edge exists between two nodes u and v if and only if $|uv| \leq 1$ where $|uv|$ is the Euclidean distance between u and v. This means that two nodes u and v are connected with an edge if and only if u's disk covers v and v's disk covers u.

Computing an MCDS in a unit disk graph is still NP-hard. How hard is it to construct a good approximation for MCDS in unit disk graphs? Cheng et al. [5] answered this question by presenting a polynomial-time approximation scheme.

Historical Background

The connected dominating set problem has been studied in graph theory for many years [22]. However, recently it becomes a hot topic due to its application in wireless networks for virtual backbone construction [4]. Guha and Khuller [10] gave a two-stage greedy approximation for the minimum connected dominating set in general graphs and showed that its performance ratio is $3 + \ln \Delta$ where Δ is the maximum node degree in the graph. To design a one-step greedy approximation to reach a similar performance ratio, the difficulty is to find a submodular potential function. In [21], Ruan et al. successfully designed a one step greedy approximation that reaches a better performance ratio $c + \ln \Delta$ for any $c > 2$. Du et al. [6] showed that there exits a polynomial-time approximation with a performance ratio $a(1 + \ln \Delta)$ for any $a > 1$. The importance of those works is that the potential functions used in their greedy algorithm are non-submodular and they managed to complete its theoretical performance evaluation with fresh ideas.

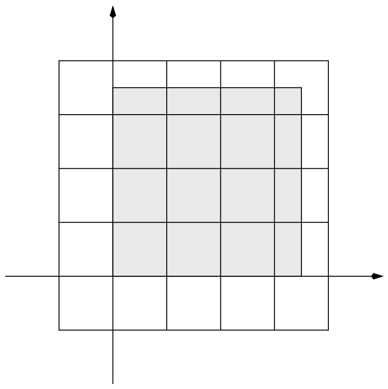
Guha and Khuller [10] also gave a negative result that there is no polynomial-time approximation with a performance ratio $\rho \ln \Delta$ for $\rho < 1$ unless $NP \subseteq DTIME(n^{O(\ln \ln n)})$. As indicated by [8], dominating sets cannot be approximated arbitrarily well, unless P almost equal to NP. These results move ones' attention from general graphs to unit disk graphs because the unit disk graph is the model for wireless sensor networks and in unit disk graphs, MCDS has a polynomial-time approximation with a constant performance ratio. While this constant ratio is getting improved step by step [1,2,19,24], Cheng et al. [5] closed this story by showing the existence of a polynomial-time approximation scheme (PTAS) for the MCDS in unit disk graphs. This means that theoretically, the performance ratio for polynomial-time approximation can be as small as $1 + \varepsilon$ for any positive number ε.

Dubhashi et al. [7] showed that once a dominating set is constructed, a connected dominating set can be easily computed in a distributed fashion. Most centralized results for dominating sets are available at [18]. In particular, a simple constant approximation for dominating sets in unit disk graphs was presented in [18]. Constant-factor approximation for minimum-weight (connected) dominating sets in UDGs was studied in [3]. A PTAS for the minimum dominating set problem in UDGs was proposed in [20]. Kuhn et al. [14] proved that a maximal independent set (MIS) (and hence also a dominating set) can be computed in asymptotically optimal time $O(\log n)$ in UDGs and a large class of bounded independence graphs. Luby [17] reported an elegant local $O(\log n)$ algorithm for MIS on general graphs. Jia et al. [11] proposed a fast $O(\log n)$ distributed approximation for dominating set in general graphs. The first constant-time distributed algorithm for dominating sets that achieves a non-trivial approximation ratio for general graphs was reported in [15]. The matching $\Omega(\log n)$ lower bound is considered to be a classic result in distributed computing [16]. For UDGs a PTAS is achievable in a distributed fashion [13]. The fastest deterministic distributed algorithm for dominating sets in UDGs was reported in [12], and the fastest randomized distributed algorithm for dominating sets in UDGs was presented in [9].

Key Results

The construction of PTAS for MCDS is based on the fact that there is a polynomial-time approximation with a constant performance ratio. Actually, this fact is quite easy to see. First, note that a unit disk contains at most five independent vertices [2]. This implies that every maximal independent set has a size at most $1 + 4opt$ where opt is the size of an MCDS. Moreover, every maximal independent set is a dominating set and it is easy to construct a maximal independent set with a spanning tree of all edges with length two. All vertices in this spanning tree form a connected dominating set of a size at most $1 + 8opt$. By improving the upper bound for the size of a maximal independent set [25] and the way to interconnecting a maximal independent set [19], the constant ratio has been improved to 6.8 with a distributed implementation.

The basic techniques in this construction is nonadaptive partition and shifting. Its general picture is as follows: First, the square containing all vertices of the input unit-disk graph is divided into a grid of small cells. Each small cell is further divided into two areas, the central area and the boundary area. The central area consists of points h distance away from the cell boundary. The boundary area consists of points within distance $h + 1$ from the boundary. Therefore, two areas are overlapping. Then a minimum union of connected dominating sets is computed in each cell for connected components of the central area of the cell. The key lemma is to



Connected Dominating Set, Figure 1
Squares Q and $\bar{Q}$

prove that the union of all such minimum unions is no more than the minimum connected dominating set for the whole graph. For vertices not in central areas, just use the part of an 8-approximation lying in boundary areas to dominate them. This part together with the above union forms a connected dominating set for the whole input unit-disk graph. By shifting the grid around to get partitions at different coordinates, a partition having the boundary part with a very small upper bound can be obtained.

The following details the construction.

Given an input connected unit-disk graph $G = (V, E)$ residing in a square $Q = \{(x, y) \mid 0 \leq x \leq q, 0 \leq y \leq q\}$ where $q \leq |V|$. To construct an approximation with a performance ratio $1 + \varepsilon$ for $\varepsilon > 0$, choose an integer $m = O((1/\varepsilon)\ln(1/\varepsilon))$. Let $p = \lfloor q/m \rfloor + 1$. Consider the square $\bar{Q} = \{(x, y) \mid -m \leq x \leq mp, -m \leq y \leq mp\}$. Partition $\bar{Q}$ into $(p + 1) \times (p + 1)$ grids so that each cell is an $m \times m$ square excluding the top and the right boundaries and hence no two cells are overlapping each other. This partition of $\bar{Q}$ is denoted by $P(0)$ (Fig. 1). In general, the partition $P(a)$ is obtained from $P(0)$ by shifting the bottom-left corner of $\bar{Q}$ from $(-m, -m)$ to $(-m + a, -m + a)$. Note that shifting from $P(0)$ to $P(a)$ for $0 \leq a \leq m$ keeps Q covered by the partition.

For each cell e (an $m \times m$ square), $C_e(d)$ denotes the set of points in e away from the boundary by distance at least d, e.g., $C_e(0)$ is the cell e itself. Denote $B_e(d) = C_e(0) - C_e(d)$. Fix a positive integer $h = 7 + 3\lfloor \log_2(4m^2/\pi) \rfloor$. Call $C_e(h)$ the *central area* of e and $B_e(h + 1)$ the *boundary area* of e. Hence the boundary area and the central area of each cell are overlapping with width one.

Central Area

Let $G_e(d)$ denote the part of input graph G lying in area $C_e(d)$. In particular, $G_e(h)$ is the part of graph G lying in the central area of e. $G_e(h)$ may consist of several connected components. Let K_e be a subset of vertices in $G_e(0)$ with a minimum cardinality such that for each connected component H of $G_e(h)$, K_e contains a connected component dominating H. In other words, K_e is a minimum union of connected dominating sets in $G(0)$ for the connected components of $G_e(h)$.

Now, denote by $K(a)$ the union of K_e for e over all cells in partition $P(a)$. $K(a)$ has two important properties:

Lemma 1 *$K(a)$ can be computed in time $n^{O(m^2)}$.*

Lemma 2 *$|K^a| \leq opt$ for $0 \leq a \leq m - 1$.*

Lemma 1 is not hard to see. Note that in a square with edge length $\sqrt{2}/2$, all vertices induce a complete subgraph in which any vertex must dominate all other vertices. It follows that the minimum dominating set for the vertices of $G_e(0)$ has size at most $(\lceil \sqrt{2}m \rceil)^2$. Hence, the size of K_e is at most $3(\lceil \sqrt{2}m \rceil)^2$ because any dominating set in a connected graph has a spanning tree with an edge length at most three. Suppose cell $G_e(0)$ has n_e vertices. Then the number of candidates for K_e is at most

$$\sum_{k=0}^{3(\lceil \sqrt{2}m \rceil)^2} \binom{n_e}{k} = n_e^{O(m^2)} .$$

Hence, computing $K(a)$ can be done in time

$$\sum_e n_e^{O(m^2)} \leq \left(\sum_e n_e \right)^{O(m^2)} = n^{O(m^2)} .$$

However, the proof of Lemma 2 is quite tedious. The reader who is interested in it may find it in [5].

Boundary Area

Let F be a connected dominating set of G satisfying $|F| \leq 8opt + 1$. Denote by $F(a)$ the subset of F lying in the boundary area $B_a(h + 1)$. Since F is constructed in polynomial-time, only the size of $F(a)$ needs to be studied.

Lemma 3 *Suppose $h = 7 + 3\lfloor \log_2(4m^2/\pi) \rfloor$ and $\lfloor m/(h + 1) \rfloor \geq 32/\varepsilon$. Then there is at least half of $i = 0, 1, \ldots, \lfloor m/(h+1) \rfloor - 1$ such that $|F(i(h + 1))| \leq \varepsilon \cdot opt$.*

Proof Let $F_H(a)$ ($F_V(a)$) denote the subset of vertices in $F(a)$ each with distance $< h + 1$ from the horizontal (vertical) boundary of some cell in $P(a)$. Then

$F(a) = F_H(a) \cup F_V(a)$. Moreover, all $F_H(i(h+1))$ for $i = 0, 1, \ldots, \lfloor m/(h+1) \rfloor - 1$ are disjoint. Hence,

$$\sum_{i=0}^{\lfloor m/(h+1)\rfloor - 1} |F_H(i(h+1))| \le |F| \le 8opt.$$

Similarly, all $F_V(i(h+1))$ for $i = 0, 1, \ldots, \lfloor m/(h+1) \rfloor - 1$ are disjoint and

$$\sum_{i=0}^{\lfloor m/(h+1)\rfloor - 1} |F_V(i(h+1))| \le |F| \le 8opt\,.$$

Thus

$$\begin{aligned}&\sum_{i=0}^{\lfloor m/(h+1)\rfloor - 1} |F(i(h+1))|\\ \le &\sum_{i=0}^{\lfloor m/(h+1)\rfloor - 1} (|F_H(i(h+1))| + |F_V(i(h+1))|)\\ \le\ & 16opt\,.\end{aligned}$$

That is,

$$\frac{1}{\lfloor m/(h+1)\rfloor} \sum_{i=0}^{\lfloor m/(h+1)\rfloor - 1} |F(i(h+1))| \le (\varepsilon/2)opt.$$

This means that there are at least half of $F(i(h+1))$ for $i = 0, 1, \lfloor m/(h+1) \rfloor - 1$ satisfying

$$|F(i(h+1))| \le \varepsilon \cdot opt\,. \qquad \square$$

Putting Together

Now put $K(a)$ and $F(a)$. By Lemmas 2 and 3, there exists $a \in \{0, h+1, \ldots, (\lfloor m/(h+1)\rfloor - 1)(h+1)\}$ such that

$$|K(a) \cup F(a)| \le (1+\varepsilon)opt.$$

Lemma 4 *For $0 \le a \le m-1$, $K(a) \cup F(a)$ is a connected dominating for input connected graph G.*

Proof $K(a) \cup F(a)$ is clearly a dominating set for input graph G. Its connectivity can be shown as follows. Note that the central area and the boundary area are overlapping with an area of width one. Thus, for any connected component H of the subgraph $G_e(h)$, $F(a)$ has a vertex in H. Hence, $F(a)$ must connect to any connected dominating set for H, especially, the one D_H in $K(a)$. This means that D_H has making up the connections of F lost from cutting a part in H. Therefore, the connectivity of $K(a) \cup F(a)$ follows from the connectivity of F. $\square$

By summarizing the above results, the following result is obtained:

Theorem 1 *There is a $(1+\varepsilon)$-approximation for MCDS in connected unit-disk graphs, running in time $n^{O((1/\varepsilon)\log(1/\varepsilon)^2)}$.*

Applications

An important application of connected dominating sets is to construct virtual backbones for wireless networks, especially, wireless sensor networks [4]. The topology of a wireless sensor network is often a unit disk graph.

Open Problems

In general, the topology of a wireless network is a disk graph, that is, each vertex is associated with a disk. Different disks may have different sizes. There is an edge from vertex u to vertex v if and only if the disk at u covers v. A virtual backbone in disk graphs is a subset of vertices, which induces a strongly connected subgraph, such that every vertex not in the subset has an in-edge coming from a vertex in the subset and also has an out-edge going into a vertex in the subset. Such a virtual backbone can be considered as a *connected dominating set* in disk graph. Is there a polynomial-time approximation with a constant performance ratio? It is open right now. Thai et al. [23] has made some effort towards this direction.

Cross References

- ▶ Dominating Set
- ▶ Exact Algorithms for Dominating Set
- ▶ Greedy Set-Cover Algorithms
- ▶ Max Leaf Spanning Tree

Recommended Reading

1. Alzoubi, K.M., Wan, P.-J., Frieder, O.: Message-optimal connected dominating sets in mobile ad hoc networks. In: ACM MOBIHOC, Lausanne, Switzerland, 09–11 June 2002
2. Alzoubi, K.M., P.-J.Wan, Frieder, O.: New Distributed Algorithm for Connected Dominating Set in Wireless Ad Hoc Networks. In: HICSS35, Hawaii, January 2002
3. Ambuhl, C., Erlebach, T., Mihalak, M., Nunkesser, M.: Constant-Factor Approximation for Minimum-Weight (Connected) Dominating Sets in Unit Disk Graphs. In: LNCS, vol. 4110, pp 3–14. Springer, Berlin (2006)
4. Blum, J., Ding, M., Thaeler, A., Cheng, X.: Applications of Connected Dominating Sets in Wireless Networks. In: Du, D.-Z., Pardalos, P. (eds.) Handbook of Combinatorial Optimization, pp. 329–369. Kluwer Academic (2004)
5. Cheng, X., Huang, X., Li, D., Wu, W., Du, D.-Z.: A polynomial-time approximation scheme for minimum connected dominating set in ad hoc wireless networks. Networks **42**, 202–208 (2003)

6. Du, D.-Z., Graham, R.L., Pardalos, P.M., Wan, P.-J., Wu, W., Zhao, W.: Analysis of greedy approximations with nonsubmodular potential functions. In: Proceedings of the 19th annual ACM-SIAM Symposium on Discrete Algorithms (SODA) pp. 167–175. January 2008
7. Dubhashi, D., Mei, A., Panconesi, A., Radhakrishnan, J., Srinivasan, A.: Fast Distributed Algorithms for (Weakly) Connected Dominating Sets and Linear-Size Skeletons. In: SODA, 2003, pp. 717–724
8. Feige, U.: A Threshold of ln n for Approximating Set Cover. J. ACM **45**(4) 634–652 (1998)
9. Gfeller, B., Vicari, E.: A Randomized Distributed Algorithm for the Maximal Independent Set Problem in Growth-Bounded Graphs. In: PODC 2007
10. Guha, S., Khuller, S.: Approximation algorithms for connected dominating sets. Algorithmica **20**, 374–387 (1998)
11. Jia, L., Rajaraman, R., Suel, R.: An Efficient Distributed Algorithm for Constructing Small Dominating Sets. In: PODC, Newport, Rhode Island, USA, August 2001
12. Kuhn, F., Moscibroda, T., Nieberg, T., Wattenhofer, R.: Fast Deterministic Distributed Maximal Independent Set Computation on Growth-Bounded Graphs. In: DISC, Cracow, Poland, September 2005
13. Kuhn, F., Moscibroda, T., Nieberg, T., Wattenhofer, R.: Local Approximation Schemes for Ad Hoc and Sensor Networks. In: DIALM-POMC, Cologne, Germany, September 2005
14. Kuhn, F., Moscibroda, T., Wattenhofer, R.: On the Locality of Bounded Growth. In: PODC, Las Vegas, Nevada, USA, July 2005
15. Kuhn, F., Wattenhofer, R.: Constant-Time Distributed Dominating Set Approximation. In: PODC, Boston, Massachusetts, USA, July 2003
16. Linial, N.: Locality in distributed graph algorithms. SIAM J. Comput. **21**(1), 193–201 (1992)
17. Luby, M.: A Simple Parallel Algorithm for the Maximal Independent Set Problem. SIAM J. Comput. **15**, 1036–1053 (1986)
18. Marathe, M.V., Breu, H., Hunt III, H.B., Ravi, S.S., Rosenkrantz, D.J.: Simple Heuristics for Unit Disk Graphs. Networks **25**, 59–68 (1995)
19. Min, M., Du, H., Jia, X., Huang, X., Huang, C.-H., Wu, W.: Improving construction for connected dominating set with Steiner tree in wireless sensor networks. J. Glob. Optim. **35**, 111–119 (2006)
20. Nieberg, T., Hurink, J.L.: A PTAS for the Minimum Dominating Set Problem in Unit Disk Graphs. LNCS, vol. 3879, pp. 296–306. Springer, Berlin (2006)
21. Ruan, L., Du, H., Jia, X., Wu, W., Li, Y., Ko, K.-I.: A greedy approximation for minimum connected dominating set. Theor. Comput. Sci. **329**, 325–330 (2004)
22. Sampathkumar, E., Walikar, H.B.: The Connected Domination Number of a Graph. J. Math. Phys. Sci. **13**, 607–613 (1979)
23. Thai, M.T., Wang F., Liu, D., Zhu, S., Du, D.-Z.: Connected Dominating Sets in Wireless Networks with Different Transmission Range. IEEE Trans. Mob. Comput. **6**(7), 721–730 (2007)
24. Wan, P.-J., Alzoubi, K.M., Frieder, O.: Distributed Construction of Connected Dominating Set in Wireless Ad Hoc Networks. In: IEEE INFOCOM 2002
25. Wu, W., Du, H., Jia, X., Li, Y., Huang, C.-H.: Minimum Connected Dominating Sets and Maximal Independent Sets in Unit Disk Graphs. Theor. Comput. Sci. **352**, 1–7 (2006)

Connectivity and Fault-Tolerance in Random Regular Graphs
2000; Nikoletseas, Palem, Spirakis, Yung

SOTIRIS NIKOLETSEAS
Department of Computer Engineering and Informatics, Computer Technology Institute, University of Patras and CTI, Patras, Greece

Keywords and Synonyms

Robustness

Problem Definition

A new model of random graphs was introduced in [7], that of random regular graphs with edge faults (denoted hereafter by $G^r_{n,p}$), obtained by selecting the edges of a random member of the set of all regular graphs of degree r independently and with probability p. Such graphs can represent a communication network in which the links fail independently and with probability $f = 1 - p$. A formal definition of the probability space $G^r_{n,p}$ follows.

Definition 1 (the $G^r_{n,p}$ probability space) Let G^r_n be the probability space of all random regular graphs with n vertices where the degree of each vertex is r. The probability space $G^r_{n,p}$ of random regular graphs with edge faults is constructed by the following two subsequent random experiments: first, a random regular graph is chosen from the space G^r_n and, second, each edge is randomly and independently deleted from this graph with probability $f = 1 - p$.

Important connectivity properties of $G^r_{n,p}$ are investigated in this entry by estimating the ranges of r, f for which, with high probability, $G^r_{n,p}$ graphs a) are highly connected b) become disconnected and c) admit a giant (i. e. of $\Theta(n)$ size) connected component of small diameter.

Notation The terms "almost certainly" (a.c.) and "with high probability" (w.h.p.) will be frequently used with their standard meaning for random graph properties. A property defined in a random graph holds almost certainly when its probability tends to 1 as the independent variable (usually the number of vertices in the graph) tends to infinity. "With high probability" means that the probability of a property of the random graph (or the success probability of a randomized algorithm) is at least $1 - n^{-\alpha}$, where $\alpha > 0$ is a constant and n is the number of vertices in the graph.

The interested reader can further study [1] for an excellent exposition of the Probabilistic Method and its applications, [2] for a classic book on random graphs, as well as [6], an excellent book on the design and analysis of randomized algorithms.

Key Results

Summary This entry studies several important connectivity properties of random regular graphs with edge faults. In order to deal with the $G^r_{n,p}$ model, [7] first extends the notion of configurations and the translation lemma between configurations and random regular graphs provided by B. Bollobás [2,3], by introducing the concept of *random configurations* to account for edge faults, and by also providing an *extended translation lemma* between random configurations and random regular graphs with edge faults.

For this new model of random regular graphs with edge faults [7] shows that:

1. For all failure probabilities $f = 1 - p \le n^{-\epsilon}$ ($\epsilon \ge \frac{3}{2r}$ fixed) and any $r \ge 3$ the biggest part of $G^r_{n,p}$ (i. e. the whole graph except of $O(1)$ vertices) remains connected and this connected part can not be separated, almost certainly, unless more than r vertices are removed. Note interestingly that the situation for this range of f and r is very similar, despite the faults, to the properties of G^r_n which is r-connected for $r \ge 3$.
2. $G^r_{n,p}$ is *disconnected* a.c. for constant f and any $r = o(\log n)$, but is *highly connected*, almost certainly, when $r \ge \alpha \log n$, where $\alpha > 0$ an appropriate constant.
3. Even when $G^r_{n,p}$ becomes disconnected, it still has *a giant component of small diameter*, even when $r = O(1)$. An $O(n \log n)$-time algorithm to construct a giant component is provided.

Configurations and Translation Lemmata

Note that it is not as easy (from the technical point of view) as in the $G_{n,p}$ case to argue about random regular graphs, because of the stochastic dependencies on the existence of the edges due to regularity. The following notion of *configurations* was introduced by B. Bollobás [2,3] to translate statements for random regular graphs to statements for the corresponding configurations which avoid the edge dependencies due to regularity and thus are much easier to deal with:

Definition 2 (Bollobás, [3]) Let $w = \cup^n_{j=1} w_j$ be a fixed set of $2m = \sum^n_{j=1} d_j$ labeled vertices where $|w_j| = d_j$. A configuration F is a partition of w into m pairs of vertices, called edges of F.

Given a configuration F, let $\theta(F)$ be the (multi)graph with vertex set V in which (i, j) is an edge if and only if F has a pair (edge) with one element in w_i and the other in w_j. Note that every regular graph $G \in G^r_n$ is of the form $\theta(F)$ for exactly $(r!)^n$ configurations. However not every configuration F with $d_j = r$ for all j corresponds to a $G \in G^r_n$ since F may have an edge entirely in some w_j or parallel edges joining w_i and w_j.

Let φ be the set of all configurations F and let G^r_n be the set of all regular graphs. Given a property (set) $Q \subseteq G^r_n$ let $Q^* \subseteq \phi$ such that $Q^* \cap \theta^{-1}(G^r_n) = \theta^{-1}(Q)$. By estimating the probability of possible cycles of length one (selfloops) and two (loops) among pairs w_i, w_j in $\theta(F)$, The following important lemma follows:

Lemma 1 (Bollobás, [2]) *If $r \ge 2$ is fixed and property Q^* holds for a.e. configuration, then property Q holds for a.e. r—regular graph.*

The main importance of the above lemma is that when studying random regular graphs, instead of considering the set of all random regular graphs, one can study the (much more easier to deal with) set of configurations.

In order to deal with edge failures, [7] introduces here the following extension of the notion of configurations:

Definition 3 (random configurations) Let $w = \cup^n_{j=1} w_j$ be a fixed set of $2m = \sum^n_{j=1} d_j$ labeled "vertices" where $|w_j| = d_j$. Let F be any configuration of the set φ. For each edge of F, remove it with probability $1 - p$, independently. Let $\hat{\phi}$ be the new set of objects and $\hat{F}$ the outcome of the experiment. $\hat{F}$ is called a *random configuration*.

By introducing probability p in every edge, an extension of the proof of Lemma 1 leads (since in both $\bar{Q}$ and $\hat{Q}$ each edge has the same probability and independence to be deleted, thus the modified spaces follow the properties of Q and Q^*) to the following extension to random configurations.

Lemma 2 (extended translation lemma) *Let $r \ge 2$ fixed and $\bar{Q}$ be a property for $G^r_{n,p}$ graphs. If $\hat{Q}$ holds for a.e. random configuration, then the corresponding property $\bar{Q}$ holds for a.e. graph in $G^r_{n,p}$.*

Multiconnectivity Properties of $G^r_{n,p}$

The case of constant link failure probability f is studied, which represents a worst case for connectivity preservation. Still, [7] shows that logarithmic degrees suffice to guarantee that $G^r_{n,p}$ remains w.h.p. highly connected, despite these constant edge failures. More specifically:

Theorem 3 *Let G be an instance of $G^r_{n,p}$ where $p = \Theta(1)$ and $r \ge \alpha \log n$, where $\alpha > 0$ an appropriate constant.*

Then G is almost certainly k-connected, where

$$k = O\left(\frac{\log n}{\log\log n}\right).$$

The proof of the above Theorem uses Chernoff bounds to estimate the vertex degrees in $G^r_{n,p}$, and "similarity" of $G^r_{n,p}$ and $G_{n,p'}$ (whose properties are known) for a suitably chosen p'.

Now the (more practical) case in which $f = 1 - p = o(1)$ is considered and it is proved that the desired connectivity properties of random regular graphs are almost preserved despite the link failures. More specifically:

Theorem 4 *Let $r \geq 3$ and $f = 1 - p = O(n^{-\epsilon})$ for $\epsilon \geq \frac{3}{2r}$. Then the biggest part of $G^r_{n,p}$ (i. e. the whole graph except of O(1) vertices) remains connected and this connected part (excluding the vertices that were originally neighbors of the O(1)-sized disconnected set) can not be separated unless more than r vertices are removed, with probability tending to 1 as n tends to $+\infty$.*

The proof is carefully extending, in the case of faults, a known technique for random regular graphs about not admitting small separators.

$G^r_{n,p}$ Becomes Disconnected

Next remark that a constant link failure probability dramatically alters the connectivity structure of the regular graph in the case of low degrees. In particular, by using the notion of random configurations, [7] proves the following theorem:

Theorem 5 *When $2 \leq r \leq \frac{\sqrt{\log n}}{2}$ and $p = \Theta(1)$ then $G^r_{n,p}$ has at least one isolated node with probability at least $1 - n^{-k}$, $k \geq 2$.*

The regime for disconnection is in fact larger, since [7] shows that $G^r_{n,p}$ is a.c. disconnected even for any $r = o(\log n)$ and constant f. The proof of this last claim is complicated by the fact that due to the range for r one has to avoid using the extended translation lemma.

Existence of a Giant Component in $G^r_{n,p}$

Since $G^r_{n,p}$ is a.c. disconnected for $r = o(\log n)$ and $1 - p = f = \Theta(1)$, it would be interesting to know whether at least a large part of the network represented by $G^r_{n,p}$ is still connected, i. e. whether the biggest connected component of $G^r_{n,p}$ is large. In particular, [7] shows that:

Theorem 6 *When $f < 1 - \frac{32}{r}$ then $G^r_{n,p}$ admits a giant (i. e. $\Theta(n)$-sized) connected component for any $r \geq 64$ with probability at least $1 - O(\log^2 n)/(n^{\alpha/3})$, where $\alpha > 0$ a constant that can be selected.*

In fact, the proof of the existence of the component includes first proving the existence (w.h.p.) of a sufficiently long (of logarithmic size) path as a basis for a BFS process starting from the vertices of that path that creates the component. The proof is quite complex: occupancy arguments are used (bins correspond to the vertices of the graphs while balls correspond to its edges); however, the random variables involved are not independent, and in order to use Chernoff-Hoeffding bounds for concentration one must prove that these random variables, although not independent, are negatively associated. Furthermore, the evaluation of the success of the BFS process uses a careful, detailed average case analysis.

The path construction and the BFS process can be viewed as an algorithm that (in case of no failures) actually reveals a giant connected component. This algorithm is very efficient, as shown by the following result:

Theorem 7 *A giant component of $G^r_{n,p}$ can be constructed in $O(n \log n)$ time, with probability at least $1 - O(\log^2 n)/(n^{\alpha/3})$, where $\alpha > 0$ a constant that can be selected.*

Applications

In recent years the development and use of distributed systems and communication networks has increased dramatically. In addition, state-of-the-art multiprocessor architectures compute over structured, regular interconnection networks. In such environments, several applications may share the same network while executing concurrently. This may lead to unavailability of certain network resources (e. g. links) for certain applications. Similarly, faults may cause unavailability of links or nodes. The aspect of *reliable distributed computing* (which means computing with the available resources and resisting faults) adds value to applications developed in such environments.

When computing in the presence of faults, one cannot assume that the actual structure of the computing environment is known. Faults may happen even in execution time. In addition, what is a "faulty" or "unavailable" link for one application may in fact be the de-allocation of that link because it is assigned (e. g. by the network operation system) to another application. The problem of analyzing allocated computation or communication in a network over a *randomly assigned subnetwork* and *in the presence of faults* has a nature different from fault analysis of special, well-structured networks (e. g. hypercube), which does not deal with network aspects. The work presented in this entry addresses this interesting issue, i. e. analyzing the average case taken over a set of possible topologies and focuses on multiconnectivity and existence of giant component properties, required for reliable distributed computing in such randomly allocated unreliable environments.

The following important application of this work should be noted: multitasking in distributed memory multiprocessors is usually performed by assigning an arbitrary subnetwork (of the interconnection network) to each task (called the *computation graph*). Each parallel program may then be expressed as communicating processors over the computation graph. Note that a multiconnectivity value k of the computation graph means also that the execution of the application can tolerate up to $k-1$ *on-line additional faults*.

Open Problems

The ideas presented in [7] inspired already further interesting research. Andreas Goerdt [4] continued the work presented in a preliminary version [8] of [7] and showed the following results: if the degree r is fixed then $p = \frac{1}{r-1}$ is a threshold probability for the existence of a linear sized component in the faulty version of almost all random regular graphs. In fact, he further shows that if each edge of an *arbitrary* graph G with maximum degree bounded above by r is present with probability $p = \frac{\lambda}{r-1}$, when $\lambda < 1$, then the faulty version of G has only components whose size is at most logarithmic in the number of nodes, with high probability. His result implies some kind of optimality of random regular graphs with edge faults. Furthermore, [5,10] investigates important expansion properties of random regular graphs with edge faults, as well as [9] does in the case of fat-trees, a common type of interconnection networks. It would be also interesting to further pursue this line of research, by also investigating other combinatorial properties (and also provide efficient algorithms) for random regular graphs with edge faults.

Cross References

► Hamilton Cycles in Random Intersection Graphs
► Independent Sets in Random Intersection Graphs
► Minimum k-Connected Geometric Networks

Recommended Reading

1. Alon, N., Spencer, J.: The Probabilistic Method. Wiley (1992)
2. Bollobás, B.: Random Graphs. Academic Press (1985)
3. Bollobás, B.: A probabilistic proof of an asymptotic formula for the number of labeled regular graphs. Eur. J. Comb. **1**, 311–316 (1980)
4. Goerdt, A.: The giant component threshold for random regular graphs with edge faults. In: Proceedings of Mathematical Foundations of Computer Science '97 (MFCS'97), pp. 279–288. (1997)
5. Goerdt, A.: Random regular graphs with edge faults: Expansion through cores. Theor. Comput. Sci. **264**(1), 91–125 (2001)
6. Motwani, R., Raghavan, P.: Randomized Algorithms. Cambridge University Press (1995)
7. Nikoletseas, S., Palem, K., Spirakis, P., Yung, M.: Connectivity Properties in Random Regular Graphs with Edge Faults. In: Special Issue on Randomized Computing of the International Journal of Foundations of Computer Science (IJFCS), vol. 11 no. 2, pp. 247–262, World Scientific Publishing Company (2000)
8. Nikoletseas, S., Palem, K., Spirakis, P., Yung, M.: Short Vertex Disjoint Paths and Multiconnectivity in Random Graphs: Reliable Network Computing. In: Proc. 21st International Colloquium on Automata, Languages and Programming (ICALP), pp. 508–515. Jerusalem (1994)
9. Nikoletseas, S., Pantziou, G., Psycharis, P., Spirakis, P.: On the reliability of fat-trees. In: Proc. 3rd International European Conference on Parallel Processing (Euro-Par), pp. 208–217, Passau, Germany (1997)
10. Nikoletseas, S., Spirakis, P.: Expander Properties in Random Regular Graphs with Edge Faults. In: Proc. 12th Annual Symposium on Theoretical Aspects of Computer Science (STACS), pp.421–432, München (1995)

Consensus with Partial Synchrony

1988; Dwork, Lynch, Stockmeyer

BERNADETTE CHARRON-BOST[1], ANDRÉ SCHIPER[2]
[1] Laboratory for Informatics, The Polytechnic School, Palaiseau, France
[2] EPFL, Lausanne, Switzerland

Keywords and Synonyms

Agreement problem

Problem Definition

Reaching agreement is one of the central issues in fault tolerant distributed computing. One version of this problem, called *Consensus*, is defined over a fixed set $\Pi = \{p_1, \dots, p_n\}$ of n processes that communicate by exchanging messages along channels. Messages are correctly transmitted (no duplication, no corruption), but some of them may be lost. Processes may fail by prematurely stopping (crash), may omit to send or receive some messages (omission), or may compute erroneous values (Byzantine faults). Such processes are said to be *faulty*. Every process $p \in \Pi$ has an initial value v_p and non-faulty processes must decide irrevocably on a common value v. Moreover, if the initial values are all equal to the same value v, then the common decision value is v. The properties that define Consensus can be split into safety properties (processes decide on the same value; the decision value must be consistent with initial values) and a liveness property (processes must eventually decide).

Various Consensus algorithms have been described [6,12] to cope with any type of process failures if there is a known[1] bound on the transmission delay of messages (*communication is synchronous*) and a known bound on process relative speeds (*processes are synchronous*). In completely asynchronous systems, where there exists no bound on transmission delays and no bound on process relative speeds, Fischer, Lynch, and Paterson [8] have proved that there is no Consensus algorithm resilient to even one crash failure. The paper by Dwork, Lynch, and Stockmeyer [7] introduces the concept of *partial synchrony*, in the sense it lies between the completely synchronous and completely asynchronous cases, and shows that partial synchrony makes it possible to solve Consensus in the presence of process failures, whatever the type of failure is.

For this purpose, the paper examines the quite realistic case of asynchronous systems that behave synchronously during some "good" periods of time. Consensus algorithms designed for synchronous systems do not work in such systems since they may violate the safety properties of Consensus during a bad period, that is when the system behaves asynchronously. This leads to the following question: is it possible to design a Consensus algorithm that never violates safety conditions in an asynchronous system, while ensuring the liveness condition when some additional conditions are met?

Key Results

The paper has been the first to provide a positive and comprehensive answer to the above question. More precisely, the paper (1) defines various types of partial synchrony and introduces a new round based computational model for partially synchronous systems, (2) gives various Consensus algorithms according to the severity of failures (crash, omission, Byzantine faults with or without authentication), and (3) shows how to implement the round based computational model in each type of partial synchrony.

[1] Intuitively, "known bound" means that the bound can be "built into" the algorithm. A formal definition is given in the next section.

Partial Synchrony

Partial synchrony applies both to communications and to processes. Two definitions for *partially synchronous communications* are given: (1) for each run, there exists an upper bound Δ on communication delays, but Δ is *unknown* in the sense it depends on the run; (2) there exists an upper bound Δ on communication delays that is common for all runs (Δ is *known*), but holds only after some time T, called the *Global Stabilization Time* (*GST*) that may depend on the run (*GST* is *unknown*). Similarly, *partially synchronous processes* are defined by replacing "transmission delay of messages" by "relative process speeds" in (1) and (2) above. That is, the upper bound on relative process speed Φ is unknown, or Φ is known but holds only after some unknown time.

Basic Round Model

The paper considers a round based model: computation is divided into *rounds* of message exchange. Each round consists of a *send step*, a *receive step*, and then a *computation step*. In a send step, each process sends messages to any subset of processes. In a receive step, some subset of the messages sent to the process during the send step at the *same* round is received. In a computation step, each process executes a state transition based on its current state and the set of messages just received.

Some of the messages that are sent may not be received, i.e, some can be lost. However, the *basic round model* assumes that there is some round GSR, such that all messages sent from non faulty processes to non faulty processes at round GSR or afterward are received.

Consensus Algorithm for Benign Faults (requires $f < n/2$)

In the paper, the algorithm is only described informally (textual form). A formal expression is given by Algorithm 1: the code of each process is given round by round, and each round is specified by the send and the computation steps (the receive step is implicit). The constant f denotes the maximum number of processes that may be faulty (crash or omission). The algorithm requires $f < n/2$.

Rounds are grouped into phases, where each phase consists in four consecutive rounds. The algorithm includes the rotating coordinator strategy: each phase k is led by a unique coordinator—denoted by $coord_k$—defined as process p_i for phase $k = i(mod\ n)$. Each process p maintains a set $Proper_p$ of values that p has heard of (*proper* values), initialized to $\{v_p\}$ where v_p is p's ini-

```
 1: Initialization:
 2:     $Acceptable_p := \{v_p\}$                                        {$v_p$ is the initial value of $p$}
 3:     $Proper_p := \{v_p\}$          {All the lines for maintaining $Proper_p$ are trivial to write, and so are omitted}
 4:     $vote_p := \perp$
 5:     $Lock_p := \emptyset$

 6: Round $r = 4k - 3$:
 7:     Send:
 8:         send $\langle Acceptable_p \rangle$ to $coord_k$

 9:     Compute:
10:         if $p = coord_k$ and $p$ receives at least $\geq n - f$ messages containing a common value then
11:             $vote_p$ := select one of these common acceptable values

12: Round $r = 4k - 2$:
13:     Send:
14:         if $p = coord_k$ and $vote_p \neq \perp$ then
15:             send $\langle vote_p \rangle$ to all processes

16:     Compute:
17:         if received $\langle v \rangle$ from $coord_k$ then
18:             $Lock_p := Lock_p \setminus \{v, -\}$; $Lock_p := Lock_p \cup \{(v, k)\}$;

19: Round $r = 4k - 1$:
20:     Send:
21:         if $\exists v$ s.t. $(v, k) \in Lock_p$ then
22:             send $\langle ack \rangle$ to $coord_k$

23:     Compute:
24:         if $p = coord_k$ then
25:             if received at least $\geq f + 1$ $ack$ messages then
26:                 DECIDE($vote_p$);
27:             $vote_p := \perp$

28: Round $r = 4k$:
29:     Send:
30:         send $\langle Lock_p \rangle$ to all processes

31:     Compute:
32:         for all $(v, \theta) \in Lock_p$ do
33:             if received $(w, \overline{\theta})$ s.t. $w \neq v$ and $\overline{\theta} \geq \theta$ then      {release lock on $v$}
34:                 $Lock_p := Lock_p \cup \{(w, \overline{\theta})\} \setminus \{(v, \theta)\}$;
35:         if $|Lock_p| = 1$ then
36:             $Acceptable_p := v$ where $(v, -) \in Lock_p$
37:         else
38:             if $Lock_p = \emptyset$ then $Acceptable_p := Proper_p$ else $Acceptable_p := \emptyset$
```

Consensus with Partial Synchrony, Algorithm 1
Consensus algorithm in the basic round model for benign faults ($f < n/2$)

tial value. Process p attaches $Proper_p$ to each message it sends.

Process p may *lock* value v when p thinks that some process might decide v. Thus value v is an *acceptable* value to p if (1) v is a proper value to p, and (2) p does not have a lock on any value except possibly v (lines 35 to 38).

At the first round of phase k (round $4k - 3$), each process sends the list of its acceptable values to $coord_k$. If $coord_k$ receives at least $n - f$ sets of acceptable values that all contain some value v, then $coord_k$ votes for v (line 11), and sends its vote to all at second round $4k - 2$. Upon receiving a vote for v, any process locks v in the current phase (line 18), releases any earlier lock on v, and sends an acknowledgment to $coord_k$ at the next round $4k - 1$. If the latter process receives acknowledgments from at least $f + 1$ processes, then it decides (line 26). Finally locks are released at round $4k$—for any value v, only the lock from the most recent phase is kept, see line 34—and the set of values *acceptable* to p is updated (lines 35 to 38).

Consensus Algorithm for Byzantine Faults (requires $f < n/3$)

Two algorithms for Byzantine faults are given. The first algorithm assumes *signed messages*, which means that any process can verify the origin of all messages. This fault model is called *Byzantine faults with authentication*. The algorithm has the same phase structure as Algorithm 1. The difference is that (1) messages are signed, and (2) "proofs" are carried by some messages. A proof carried by message m sent by some process p_i in phase k consists of a set of signed messages $sgn_j(m', k)$, proving that p_i received message (m', k) in phase k from p_j before sending m. A proof is carried by the message send at line 16 and line 30 (Algorithm 1). Any process receiving a message carrying a proof accepts the message and behaves accordingly if—and only if the proof is found valid. The algorithm requires $f < n/3$ (less than a third of the processes are faulty).

The second algorithm does not assume a mechanism for signing messages. Compared to Algorithm 1, the structure of a phase is slightly changed. The problem is related to the vote sent by the coordinator (line 15). Can a Byzantine coordinator fool other processes by not sending the right vote? With signed messages, such a behavior can be detected thanks to the "proofs" carried by messages. A different mechanism is needed in the absence of signature.

The mechanism is a small variation of the *Consistent Broadcast* primitive introduced by Srikanth and Toueg [15]. The broadcast primitive ensures that (1) if a non faulty process broadcasts m, then every non faulty process delivers m, and (2) if some non faulty process delivers m, then all non faulty processes also eventually deliver m. The implementation of this broadcast primitive requires two rounds, which define a *superround*. A phase of the algorithm consists now of three superrounds. The superrounds $3k-2$, $3k-1$, $3k$ mimic rounds $4k-3$, $4k-2$, and $4k-1$ of Algorithm 1, respectively. Lock-release of phase k occurs at the end of superround $3k$, i. e., does not require an additional round, as it does in the two previous algorithms. The algorithm also requires $f < n/3$.

The Special Case of Synchronous Communication

By strengthening the round based computational model, the authors show that synchronous communication allow higher resiliency. More precisely, the paper introduces the model called the *basic round model with signals*, in which upon receiving a signal at round r, every process knows that all the non faulty processes have received the messages that it has sent during round r. At each round after *GSR*, each non faulty process is guaranteed to receive a signal. In this computational model, the authors present three new algorithms tolerating less than n benign faults, $n/2$ Byzantine faults with authentication, and $n/3$ Byzantine faults respectively.

Implementation of the Basic Round Model

The last part of the paper consists of algorithms that simulate the basic round model under various synchrony assumption, for crash faults and Byzantine faults: first with partially synchronous communication and synchronous processes (case 1), second with partially synchronous communication and processes (case 2), and finally with partially synchronous processes and synchronous communication (case 3).

In case 1, the paper first assumes the basic case $\Phi = 1$, i. e., all non faulty process progress exactly at the same speed, which means that they have a common notion of time. Simulating the basic round model is simple in this case. In case 2 processes do not have a common notion of time. The authors handle this case by designing an algorithm for clock synchronization. Then each process uses its private clock to determine its current round. So processes alternate between steps of the clock synchronization algorithm and steps simulating rounds of the basic round model. With synchronous communication (case 3), the authors show that for any type of faults, the so-called basic round model with signals is implementable.

Note that, from the very definition of partial synchrony, the six algorithms share the fundamental property of tolerating message losses, provided they occur during a finite period of time.

Upper Bound for Resiliency

In parallel, the authors exhibit upper bounds for the resiliency degree of Consensus algorithms in each partially synchronous model, according to the type of faults. They show that their Consensus algorithms achieve these upper bounds, and so are optimal with respect to their resiliency degree. These results are summarized in Table 1.

Applications

Availability is one of the key features of critical systems, and is defined as the ratio of the time the system is operational over the total elapsed time. Availability of a system can be increased by replicating its critical components. Two main classes of replication techniques have been considered: *active* replication and *passive* replication. The Consensus problem is at the heart of the implementation of these replication techniques. For example, active replication, also called *state machine replication* [10,14], can be implemented using the group communication primitive called *Atomic Broadcast*, which can be reduced to Consensus [3].

Agreement needs also to be reached in the context of distributed transactions. Indeed, all participants of a distributed transaction need to agree on the output *commit* or *abort* of the transaction. This agreement problem, called *Atomic Commitment*, differs from Consensus in the validity property that connects decision values (*commit* or *abort*) to the initial values (favorable to commit, or de-

Consensus with Partial Synchrony, Table 1
Tight resiliency upper bounds (P stands for "process", C for "communication"; 0 means "asynchronous", 1/2 means "partially synchronous", and 1 means "synchronous")

	$P=0$ $C=0$	$P=1/2$ $C=1/2$	$P=1$ $C=1/2$	$P=1/2$ $C=1$	$P=1$ $C=1$
Benign	0	$\lceil(n-1)/2\rceil$	$\lceil(n-1)/2\rceil$	$n-1$	$n-1$
Authenticated Byzantine	0	$\lceil(n-1)/3\rceil$	$\lceil(n-1)/3\rceil$	$\lceil(n-1)/2\rceil$	$n-1$
Byzantine	0	$\lceil(n-1)/3\rceil$	$\lceil(n-1)/3\rceil$	$\lceil(n-1)/3\rceil$	$\lceil(n-1)/3\rceil$

manding abort) [9]. In the case decisions are required in all executions, the problem can be reduced to Consensus if the abort decision is acceptable although all processes were favorable to commit, in some restricted failure cases.

Open Problems

A slight modification to each of the algorithms given in the paper is to force a process repeatedly to broadcast the message "Decide v" after it decides v. Then the resulting algorithms share the property that all non faulty processes definitely make a decision within $O(f)$ rounds after GSR, and the constant factor varies between 4 (benign faults) and 12 (Byzantine faults). A question raised by the authors at the end of the paper is whether this constant can be reduced. Interestingly, a positive answer has been given later, in the case of benign faults and $f < n/3$, with a constant factor of 2 instead of 4. This can be achieved with deterministic algorithms, see [4], based on the communication schema of the Rabin randomized Consensus algorithm [13].

The second problem left open is the generalization of this algorithmic approach—namely, the design of algorithms that are always safe and that terminate when a sufficiently long good period occurs—to other fault tolerant distributed problems in partially synchronous systems. The latter point has been addressed for the Atomic Commitment and Atomic Broadcast problems (see Sect. "Applications").

Cross References

► Asynchronous Consensus Impossibility
► Failure Detectors
► Randomization in Distributed Computing

Recommended Reading

1. Bar-Noy, A., Dolev, D., Dwork, C., Strong, H.R.: Shifting Gears: Changing Algorithms on the Fly To Expedite Byzantine Agreement. In: PODC, 1987, pp. 42–51
2. Chandra, T.D., Hadzilacos, V., Toueg, S.: The Weakest Failure Detector for Solving Consensus. J. ACM **43**(4), 685–722 (1996)
3. Chandra, T.D., Toueg, S.: Unreliable failure detectors for reliable distributed systems. J. ACM **43**(2), 225–267 (1996)
4. Charron-Bost, B., Schiper A.: The "Heard-Of" model: Computing in distributed systems with benign failures. Technical Report, EPFL (2007)
5. Dolev, D., Dwork, C., Stockmeyer, L.: On the minimal synchrony needed for distributed consensus. J. ACM **34**(1), 77–97 (1987)
6. Dolev, D., Strong, H.R.: Authenticated Algorithms for Byzantine Agreement. SIAM J. Comput. **12**(4), 656–666 (1983)
7. Dwork, C., Lynch, N., Stockmeyer, L.: Consensus in the presence of partial synchrony. J. ACM **35**(2), 288–323 (1988)
8. Fischer, M., Lynch, N., Paterson, M.: Impossibility of Distributed Consensus with One Faulty Process. J. ACM **32**, 374–382 (1985)
9. Gray, J.: A Comparison of the Byzantine Agreement Problem and the Transaction Commit Problem. In: Fault-Tolerant Distributed Computing [Asilomar Workshop 1986]. LNCS, vol. 448, pp. 10–17. Springer, Berlin (1990)
10. Lamport, L.: Time, Clocks, and the Ordering of Events in a Distributed System. Commun. ACM **21**(7), 558–565 (1978)
11. Lamport, L.: The Part-Time Parliament. ACM Trans. on Computer Systems **16**(2), 133–169 (1998)
12. Pease, M.C., Shostak, R.E., Lamport, L.: Reaching Agreement in the Presence of Faults. J. ACM **27**(2), 228–234 (1980)
13. Rabin, M.: Randomized Byzantine Generals. In: Proc. 24th Annual ACM Symposium on Foundations of Computer Science, 1983, pp. 403–409
14. Schneider, F.B.: Replication Management using the State-Machine Approach. In Sape Mullender, editor, Distributed Systems, pp. 169–197. ACM Press (1993)
15. Srikanth, T.K., Toueg, S.: Simulating Authenticated Broadcasts to Derive Simple Fault-Tolerant Algorithms. Distrib. Comp. **2**(2), 80–94 (1987)

Constructing a Galled Phylogenetic Network

2006; Jansson, Nguyen, Sung

WING-KIN SUNG
Department of Computer Science, National University of Singapore, Singapore, Singapore

Keywords and Synonyms

Topology with independent recombination events; Galled-tree; Gt-network; Level-1 phylogenetic network

Problem Definition

A *phylogenetic tree* is a binary, rooted, unordered tree whose leaves are distinctly labeled. A *phylogenetic network* is a generalization of a phylogenetic tree formally defined as a rooted, connected, directed acyclic graph in which: (1) each node has outdegree at most 2; (2) each node has indegree 1 or 2, except the root node, which has indegree 0; (3) no node has both indegree 1 and outdegree 1; and (4) all nodes with outdegree 0 are labeled by elements from a finite set L in such a way that no two nodes are assigned the same label. Nodes of outdegree 0 are referred to as *leaves* and identified with their corresponding elements in L. For any phylogenetic network N, let $\mathcal{U}(N)$ be the undirected graph obtained from N by replacing each directed edge by an undirected edge. N is said to be a *galled phylogenetic network* (*galled network* for short) if all cycles in $\mathcal{U}(N)$ are node-disjoint. Galled networks are also known in the literature as *topologies with independent recombination events* [17], *galled trees* [3], *gt-networks* [13], and *level-1 phylogenetic networks* [2,7].

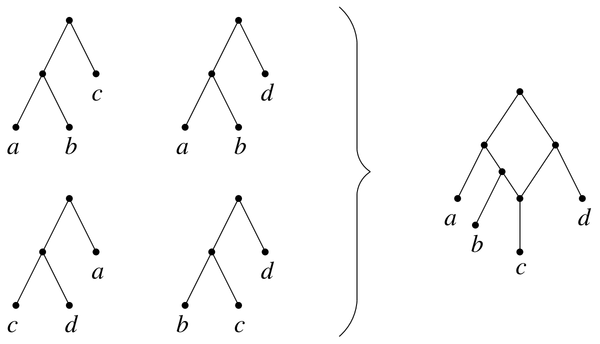
A phylogenetic tree with exactly three leaves is called a *rooted triplet*. The unique rooted triplet on a leaf set $\{x, y, z\}$ in which the lowest common ancestor of x and y is a proper descendant of the lowest common ancestor of x and z (or, equivalently, where the lowest common ancestor of x and y is a proper descendant of the lowest common ancestor of y and z) is denoted by $(\{x, y\}, z)$. For any phylogenetic network N, a rooted triplet t is said to be *consistent* with N if t is an induced subgraph of N, and a set $\mathcal{T}$ of rooted triplets is *consistent* with N if every rooted triplet in $\mathcal{T}$ is consistent with N.

Denote the set of leaves in any phylogenetic network N by $\Lambda(N)$, and for any set $\mathcal{T}$ of rooted triplets, define $\Lambda(\mathcal{T}) = \bigcup_{t_i \in \mathcal{T}} \Lambda(t_i)$. A set $\mathcal{T}$ of rooted triplets is *dense* if for each $\{x, y, z\} \subseteq \Lambda(\mathcal{T})$ at least one of the three possible rooted triplets $(\{x, y\}, z)$, $(\{x, z\}, y)$, and $(\{y, z\}, x)$ belongs to $\mathcal{T}$. If $\mathcal{T}$ is dense, then $|\mathcal{T}| = \Theta(|\Lambda(\mathcal{T})|^3)$. Furthermore, for any set $\mathcal{T}$ of rooted triplets and $L' \subseteq \Lambda(\mathcal{T})$, define $\mathcal{T} \mid L'$ as the subset of $\mathcal{T}$ consisting of all rooted triplets t with $\Lambda(t) \subseteq L'$. The problem [8] considered here is as follows.

Problem 1 *Given a set $\mathcal{T}$ of rooted triplets, output a galled network N with $\Lambda(N) = \Lambda(\mathcal{T})$ such that N and $\mathcal{T}$ are consistent, if such a network exists; otherwise, output null. (See Fig. 1 for an example.)*

Another related problem is the forbidden triplet problem [4]. It is defined as follows.

Problem 2 *Given two sets $\mathcal{T}$ and $\mathcal{F}$ of rooted triplets, a galled network N $\Lambda(N) = \Lambda(\mathcal{T})$ such that (1) N and $\mathcal{T}$ are consistent and (2) every rooted triplet in $\mathcal{F}$ is not consistent with N. If such a network N exists, it is to be reported; otherwise, output null.*



Constructing a Galled Phylogenetic Network, Figure 1
A dense set $\mathcal{T}$ of rooted triplets with leaf set $\{a, b, c, d\}$ and a galled phylogenetic network which is consistent with $\mathcal{T}$. Note that this solution is not unique

Below, write $L = \Lambda(\mathcal{T})$ and $n = |L|$.

Key Results

Theorem 1 *Given a dense set $\mathcal{T}$ of rooted triplets with leaf set L, a galled network consistent with $\mathcal{T}$ in $O(n^3)$ time can be reported, where $n = |L|$.*

Theorem 2 *Given a nondense set $\mathcal{T}$ of rooted triplets, it is NP-hard to determine if there exists a galled network that is consistent with $\mathcal{T}$. Also, it is NP-hard to determine if there exists a simple phylogenetic network that is consistent with $\mathcal{T}$.*

Below, the problem of returning a galled network N consistent with the maximum number of rooted triplets in $\mathcal{T}$ for any (not necessarily dense) $\mathcal{T}$ is considered. Since Theorem 2 implies that this problem is NP-hard, approximation algorithms are studied. An algorithm is called k-approximable if it always returns a galled network N such that $N(\mathcal{T})/|\mathcal{T}| \geq k$, where $N(\mathcal{T})$ is the number of rooted triplets in $\mathcal{T}$ that are consistent with N.

Theorem 3 *Given a set of rooted triplets $\mathcal{T}$, there is no approximation algorithm that infers a galled network N such that $N(\mathcal{T})/|\mathcal{T}| \geq 0.4883$.*

Theorem 4 *Given a set of rooted triplets $\mathcal{T}$, there exists an approximation algorithm for inferring a galled network N such that $N(\mathcal{T})/|\mathcal{T}| \geq 5/12$. The running time of the algorithm is $O(|\Lambda(\mathcal{T})||\mathcal{T}|^3)$.*

The next theorem considers the forbidden triplet problem.

Theorem 5 *Given two sets of rooted triplets $\mathcal{T}$ and $\mathcal{F}$, there exists an $O(|L|^2|\mathcal{T}|(|\mathcal{T}| + |\mathcal{F}|))$-time algorithm for inferring a galled network N that guarantees $|N(\mathcal{T})| - |N(\mathcal{F})| \geq 5/12(|\mathcal{T}| - |\mathcal{F}|)$.*

Applications

Phylogenetic networks are used by scientists to describe evolutionary relationships that do not fit the traditional models in which evolution is assumed to be treelike (see, e. g., [12,16]). Evolutionary events such as horizontal gene transfer or hybrid speciation (often referred to as *recombination events*) that suggest convergence between objects cannot be represented in a single tree [3,5,13,15,17] but can be modeled in a phylogenetic network as internal nodes having more than one parent. Galled networks are an important type of phylogenetic network that have attracted special attention in the literature [2,3,13,17] due to their biological significance (see [3]) and their simple, almost treelike, structure. When the number of recombination events is limited and most of the recombination events have occurred recently, a galled network may suffice to accurately describe the evolutionary process under study [3].

An open challenge in the field of phylogenetics is to develop efficient and reliable methods for constructing and comparing phylogenetic networks. For example, to construct a meaningful phylogenetic network for a large subset of the human population (which may subsequently be used to help locate regions in the genome associated with some observable trait indicating a particular disease) in the future, efficient algorithms are crucial because the input can be expected to be very large.

The motivation behind the rooted triplet approach taken in this paper is that a highly accurate tree for each cardinality three subset of a leaf set can be obtained through maximum-likelihood-based methods such as [1] or Sibley–Ahlquist-style DNA–DNA hybridization experiments (see [10]). Hence, the algorithms presented in [7] and here can be used as the merging step in a divide-and-conquer approach to constructing phylogenetic networks analogous to the quartet method paradigm for inferring unrooted phylogenetic trees [9,11] and other supertree methods (see [6,14] and references therein). Dense input sets in particular are considered since this case can be solved in polynomial time.

Open Problems

For the rooted triplet problem, the current approximation ratio is not tight ($0.4883 \geq N(\mathcal{T})/|\mathcal{T}| \geq 5/12$). It is open if a tight approximation ratio can be found for this problem. Similarly, a tight approximation ratio needs to be found for the forbidden triplet problem.

Another direction is to work on a fixed-parameter polynomial-time algorithm. Assume the number of hybrid nodes is bounded by h. Can an algorithm that is polynomial in $|\mathcal{T}|$ while exponential in h be given?

Cross References

- ▶ Directed Perfect Phylogeny (Binary Characters)
- ▶ Distance-Based Phylogeny Reconstruction (Fast-Converging)
- ▶ Distance-Based Phylogeny Reconstruction (Optimal Radius)
- ▶ Perfect Phylogeny (Bounded Number of States)
- ▶ Phylogenetic Tree Construction from a Distance Matrix

Recommended Reading

1. Chor, B., Hendy, M., Penny, D.: Analytic solutions for three-taxon ML_{MC} trees with variable rates across sites. In: Proc. 1st Workshop on Algorithms in Bioinformatics (WABI 2001). LNCS, vol. 2149, pp. 204–213. Springer, Berlin (2001)
2. Choy, C., Jansson, J., Sadakane, K., Sung, W.-K.: Computing the maximum agreement of phylogenetic networks. In: Proc. Computing: the 10th Australasian Theory Symposium (CATS 2004), 2004, pp. 33–45
3. Gusfield, D., Eddhu, S., Langley, C.: Efficient reconstruction of phylogenetic networks with constrained recombination. In: Proc. of Computational Systems Bioinformatics (CSB2003), 2003 pp. 363–374
4. He, Y.-J., Huynh, T.N.D., Jannson, J., Sung, W.-K.: Inferring phylogenetic relationships avoiding forbidden rooted triplets. J Bioinform. Comput. Biol. **4**(1), 59–74 (2006)
5. Hein, J.: Reconstructing evolution of sequences subject to recombination using parsimony. Math. Biosci. **98**(2), 185–200 (1990)
6. Henzinger, M.R., King, V., Warnow, T.: Constructing a tree from homeomorphic subtrees, with applications to computational evolutionary biology. Algorithmica **24**(1), 1–13 (1999)
7. Jansson, J., Sung, W.-K.: Inferring a level-1 phylogenetic network from a dense set of rooted triplets. In: Proc. 10th International Computing and Combinatorics Conference (COCOON 2004), 2004
8. Jansson, J., Nguyen, N.B., Sung, W.-K.: Algorithms for combining rooted triplets into a galled phylogenetic network. SIAM J. Comput. **35**(5), 1098–1121 (2006)
9. Jiang, T., Kearney, P., Li, M.: A polynomial time approximation scheme for inferring evolutionary trees from quartet topologies and its application. SIAM J. Comput. **30**(6), 1942–1961 (2001)
10. Kannan, S., Lawler, E., Warnow, T.: Determining the evolutionary tree using experiments. J. Algorithms **21**(1), 26–50 (1996)
11. Kearney, P.: Phylogenetics and the quartet method. In: Jiang, T., Xu, Y., and Zhang, M.Q. (eds.) Current Topics in Computational Molecular Biology, pp. 111–133. MIT Press, Cambridge (2002)

12. Li., W.-H.: Molecular Evolution. Sinauer, Sunderland (1997)
13. Nakhleh, L., Warnow, T., Linder, C.R.: Reconstructing reticulate evolution in species – theory and practice. In: Proc. 8th Annual International Conference on Research in Computational Molecular Biology (RECOMB 2004), 2004, pp. 337–346
14. Ng, M.P., Wormald, N.C.: Reconstruction of rooted trees from subtrees. Discrete Appl. Math. **69**(1–2), 19–31 (1996)
15. Posada, D., Crandall, K.A.: Intraspecific gene genealogies: trees grafting into networks. TRENDS Ecol. Evol. **16**(1), 37–45 (2001)
16. Setubal, J.C., Meidanis, J.: Introduction to Computational Molecular Biology. PWS, Boston (1997)
17. Wang, L., Zhang, K., Zhang, L.: Perfect phylogenetic networks with recombination. J. Comput. Biol. **8**(1), 69–78 (2001)

Coordination Ratio

► Price of Anarchy
► Selfish Unsplittable Flows: Algorithms for Pure Equilibria
► Stackelberg Games: The Price of Optimum

CPU Time Pricing

2005; Deng, Huang, Li

Li-Sha Huang
Department of Compurter Science, Tsinghua University, Beijing, China

Keywords and Synonyms

Competitive auction; Market equilibrium; Resource scheduling

Problem Definition

This problem is concerned with a Walrasian equilibrium model to determine the prices of CPU time. In a market model of a CPU job scheduling problem, the owner of the CPU processing time sells time slots to customers and the prices of each time slot depends on the seller's strategy and the customers' bids (valuation functions). In a Walrasian equilibrium, the market is clear and each customer is most satisfied according to its valuation function and current prices. The work of Deng, Huang, and Li [1] establishes the existence conditions of Walrasion equilibrium, and obtains complexity results to determine the existence of equilibrium. It also discusses the issues of excessive supply of CPU time and price dynamics.

Notations

Consider a combinatorial auction (Ω, I, V):

- Commodities: The seller sells m kinds of indivisible commodities. Let $\Omega = \{\omega_1 \times \delta_1, \dots, \omega_m \times \delta_m\}$ denote the set of commodities, where δ_j is the available quantity of the item ω_j.
- Agents: There are n agents in the market acting as buyers, denoted by $I = \{1, 2, \dots, n\}$.
- Valuation functions: Each buyer $i \in I$ has a valuation function $v_i : 2^\Omega \to \mathbb{R}^+$ to submit the maximum amount of money he is willing to pay for a certain bundle of items. Let $V = \{v_1, v_2, \dots, v_n\}$.

An XOR combination of two valuation functions v_1 and v_2 is defined by:

$$(v_1 \text{ XOR } v_2)(S) = \max\{v_1(S), v_2(S)\}$$

An *atomic bid* is a valuation function v denoted by a pair (S, q), where $S \subset \Omega$ and $q \in \mathbb{R}^+$:

$$v(T) = \begin{cases} q\,, & \text{if } S \subset T \\ 0\,, & \text{otherwise} \end{cases}$$

Any valuation function v_i can be expressed by an XOR combination of atomic bids,

$$v_i = (S_{i1}, q_{i1}) \text{ XOR } (S_{i2}, q_{i2}) \dots \text{ XOR } (S_{in}, q_{in})$$

Given (Ω, I, V) as the input, the seller will determine an *allocation* and a *price vector* as the output:

- An *allocation* $X = \{X_0, X_1, X_2, \dots, X_n\}$ is a partition of Ω, in which X_i is the bundle of commodities assigned to buyer i and X_0 is the set of unallocated commodities.
- A *price* vector p is a non-negative vector in $\mathbb{R}^m$, whose jth entry is the price of good $\omega_j \in \Omega$.

For any subset $T = \{\omega_1 \times \sigma_1, \dots, \omega_m \times \sigma_m\} \subset \Omega$, define $p(T)$ by $p(T) = \sum_{j=1}^m \sigma_j p_j$. If buyer i is assigned to a bundle X_i, his *utility* is $u_i(X_i, p) = v_i(X_i) - p(X_i)$.

Definition A *Walrasian equilibrium* for a combinatorial auction (Ω, I, V) is a tuple (X, p), where $X = \{X_0, X_1, \dots, X_n\}$ is an allocation and p is a price vector, satisfying that:

$$(1)\ p(X_0) = 0;$$

$$(2)\ u_i(X_i, p) \geq u_i(B, p), \quad \forall B \subset \Omega, \quad \forall 1 \leq i \leq n$$

Such a price vector is also called a market clearing price, or Walrasian price, or equilibrium price.

The CPU Job-Scheduling Problem

There are two types of players in a market-driven CPU resource allocation model: a resource provider and n consumers. The provider sells to the consumers CPU time slots and the consumers each have a job that requires a fixed number of CPU time, and its valuation function depends on the time slots assigned to the job, usually the last assigned CPU time slot. Assume that all jobs are released at time $t = 0$ and the ith job needs s_i time units. The jobs are interruptible without preemption cost, as is often modeled for CPU jobs.

Translating into the language of combinatorial auctions, there are m commodities (time units), $\Omega = \{\omega_1, \dots, \omega_m\}$, and n buyers (jobs) , $I = \{1, 2, \dots, n\}$, in the market. Each buyer has a valuation function v_i, which only depends on the completion time. Moreover, if not explicitly mentioned, every job's valuation function is non-increasing w.r.t. the completion time.

Key Results

Consider the following linear programming problem:

$$\max \sum_{i=1}^{n} \sum_{j=1}^{k_i} q_{ij} x_{ij}$$

$$\text{s.t.} \sum_{i,j|\omega_k \in S_{ij}} x_{ij} \le \delta_k \,, \quad \forall \omega_k \in \Omega$$

$$\sum_{j=1}^{r_i} x_{ij} \le 1 \,, \quad \forall 1 \le i \le n$$

$$0 \le x_{ij} \le 1 \,, \quad \forall i, j$$

Denote the problem by **LPR** and its integer restriction by **IP**. The following theorem shows that a non-zero gap between the integer programming problem **IP** and its linear relaxation implies the non-existence of the Walrasian equilibrium.

Theorem 1 *In a combinatorial auction, the Walrasian equilibrium exists if and only if the optimum of **IP** equals the optimum of **LPR**. The size of the LP problem is linear to the total number of XOR bids.*

Theorem 2 *Determination of the existence of Walrasian equilibrium in a CPU job scheduling problem is strong NP-hard.*

Now consider a job scheduling problem in which the customers' valuation functions are all linear. Assume n jobs are released at the time $t = 0$ for a single machine, the jth job's time span is $s_j \in \mathbb{N}^+$ and weight $w_j \ge 0$. The goal of the scheduling is to minimize the weighted completion time: $\sum_{i=1}^{n} w_i t_i$, where t_i is the completion time of job i. Such a problem is called an MWCT (Minimal Weighted Completion Time) problem.

Theorem 3 *In a single-machine MWCT job scheduling problem, Walrasian equilibrium always exists when $m \ge EM + \Delta$, where m is the total number of processor time, $EM = \sum_{i=1}^{n} s_i$ and $\Delta = \max_k \{s_k\}$. The equilibrium can be computed in polynomial time.*

The following theorem shows the existence of a non-increasing price sequence if Walrasian equilibrium exists.

Theorem 4 *If there exists a Walrasian equilibrium in a job scheduling problem, it can be adjusted to an equilibrium with consecutive allocation and a non-increasing equilibrium price vector.*

Applications

Information technology has changed people's lifestyles with the creation of many digital goods, such as word processing software, computer games, search engines, and online communities. Such a new economy has already demanded many theoretical tools (new and old, of economics and other related disciplines) be applied to their development and production, marketing, and pricing. The lack of a full understanding of the new economy is mainly due to the fact that digital goods can often be re-produced at no additional cost, though multi-fold other factors could also be part of the difficulty. The work of Deng, Huang, and Li [1] focuses on CPU time as a product for sale in the market, through the Walrasian pricing model in economics. CPU time as a commercial product is extensively studied in grid computing. Singling out CPU time pricing will help us to set aside other complicated issues caused by secondary factors, and a complete understanding of this special digital product (or service) may shed some light on the study of other goods in the digital economy.

The utilization of CPU time by multiple customers has been a crucial issue in the development of operating system concept. The rise of grid computing proposes to fully utilize computational resources, e. g. CPU time, disk space, bandwidth. Market-oriented schemes have been proposed for efficient allocation of computational grid recourses, by [2,5]. Later, various practical and simulation systems have emerged in grid resource management. Besides the resource allocation in grids, an economic mechanism has also been introduced to TCP congestion control problems, see Kelly [4].

Cross References

- ▶ Adwords Pricing
- ▶ Competitive Auction
- ▶ Incentive Compatible Selection
- ▶ Price of Anarchy

Recommended Reading

1. Deng, X., Huang, L.-S., Li, M.: On Walrasian Price of CPU time. In: Proceedings of COCOON'05, Knming, 16–19 August 2005, pp. 586–595. Algorithmica **48**(2), 159–172 (2007)
2. Ferguson, D., Yemini, Y., Nikolaou, C.: Microeconomic Algorithms for Load Balancing in Distributed Computer Systems. In: Proceedings of DCS'88, pp. 419–499. San Jose, 13–17 June 1988,
3. Goldberg, A.V., Hartline, J.D., Wright, A.: Competitive Auctions and Digital Goods. In: Proceedings of SODA'01, pp. 735–744. Washington D.C., 7–9 January 2001
4. Kelly, F.P.: Charging and rate control for elastic traffic. Eur. Trans. Telecommun. **8**, 33–37 (1997)
5. Kurose, J.F., Simha, R.: A Microeconomic Approach to Optimal Resource Allocation in Distributed Computer Systems. IEEE Trans. Comput. **38**(5), 705–717 (1989)
6. Nisan, N.: Bidding and Allocation in Combinatorial Auctions. In: Proceedings of EC'00, pp. 1–12. Minneapolis, 17–20 October 2000

Critical Range for Wireless Networks

2004; Wan, Yi

CHIH-WEI YI
Department of Computer Science,
National Chiao Tung University, Hsinchu City, Taiwan

Keywords and Synonyms

Random geometric graphs; Monotonic properties; Isolated nodes; Connectivity; Gabriel graphs; Delaunay triangulations; Greedy forward routing

Problem Definition

Given a point set V, a graph of the vertex set V in which two vertices have an edge if and only if the distance between them is at most r for some positive real number r is called a r-disk graph over the vertex set V and denoted by $G_r(V)$. If $r_1 \leq r_2$, obviously $G_{r_1}(V) \subseteq G_{r_2}(V)$. A graph property is monotonic (increasing) if a graph is with the property, then every supergraph with the same vertex set also has the property. The critical-range problem (or critical-radius problem) is concerned with the minimal range r such that $G_r(V)$ is with some monotonic property. For example, graph connectivity is monotonic and crucial to many applications. It is interesting to know whether $G_r(V)$ is connected or not. Let $\rho_{\text{con}}(V)$ denote the minimal range r such that $G_r(V)$ is connected. Then, $G_r(V)$ is connected if $r \geq \rho_{\text{con}}(V)$, and otherwise not connected. Here $\rho_{\text{con}}(V)$ is called the critical range for connectivity of V. Formally, the critical-range problem is defined as follows.

Definition 1 The critical range for a monotonic graph property π over a point set V, denoted by $\rho_\pi(V)$, is the smallest range r such that $G_r(V)$ has property π.

From another aspect, for a given geometric property, a corresponding geometric structure is usually embedded. In many cases, the critical-range problem for graph properties is related or equivalent to the longest-edge problem of corresponding geometric structures. For example, if $G_r(V)$ is connected, it contains a Euclidean minimal spanning tree (EMST), and $\rho_{\text{con}}(V)$ is equal to the largest edge length of the EMST. So the critical range for connectivity problem is equivalent to the longest edge of the EMST problem, and the critical range for connectivity is the smallest r such that $G_r(V)$ contains the EMST.

In most cases, given an instance, the critical range can be calculated by polynomial time algorithms. So it is not a hard problem to decide the critical range. Researchers are interested in the probabilistic analysis of the critical range, especially asymptotic behaviors of r-disk graphs over random point sets. Random geometric graphs [8] is a general term for the theory about r-disk graphs over random point sets.

Key Results

In the following, problems are discussed in a 2D plane. Let $X_1, X_2, \cdots$ be independent and uniformly distributed random points on a bounded region A. Given a positive integer n, the point process $\{X_1, X_2, \ldots, X_n\}$ is referred to as the uniform n-point process on A, and is denoted by $\mathcal{X}_n(A)$. Given a positive number λ, let $Po(\lambda)$ be a Poisson random variable with parameter λ, independent of $\{X_1, X_2, \ldots\}$. Then the point process $\{X_1, X_2, \ldots, X_{Po(n)}\}$ is referred to as the Poisson point process with mean n on A, and is denoted by $\mathcal{P}_n(A)$. A is called a deployment region. An event is said to be asymptotic almost sure if it occurs with a probability that converges to 1 as $n \to \infty$.

In a graph, a node is "isolated" if it has no neighbor. If a graph is connected, there exists no isolated node in the graph. The asymptotic distribution of the number of isolated nodes is given by the following theorem [2,6,14].

Theorem 1 *Let $r_n = \sqrt{\frac{\ln n + \xi}{\pi n}}$ and Ω be a unit-area disk or square. The number of isolated nodes in $G_r(\mathcal{X}_n(\Omega))$ or $G_r(\mathcal{P}_n(\Omega))$ is asymptotically Poisson with mean $e^{-\xi}$.*

According to the theorem, the probability of the event that there is no isolated node is asymptotically equal to $\exp\left(-e^{-\xi}\right)$. In the theory of random geometric graphs, if a graph has no isolated node, it is almost surely connected. Thus, the next theorem follows [6,8,9].

Theorem 2 *Let* $r_n = \sqrt{\frac{\ln n+\xi}{\pi n}}$ *and* Ω *be a unit-area disk or square. Then,*

$$\Pr\left[G_r\left(\mathcal{X}_n(\Omega)\right) \textit{ is connected}\right] \to \exp\left(-e^{-\xi}\right) , \textit{ and}$$
$$\Pr\left[G_r\left(\mathcal{P}_n(\Omega)\right) \textit{ is connected}\right] \to \exp\left(-e^{-\xi}\right) .$$

In wireless sensor networks, the deployment region is k-covered if every point in the deployment region is within the coverage ranges of at least k sensors (vertices). Assume the coverage ranges are disks of radius r centered at the vertices. Let k be a fixed non-negative integer, and Ω be the unit-area square or disk centered at the origin $\mathbf{o}$. For any real number t, let $t\Omega$ denote the set $\{tx : x \in \Omega\}$, i. e., the square or disk of area t^2 centered at the origin. Let $C_{n,r}$ (respectively, $C'_{n,r}$) denote the event that Ω is $(k+1)$-covered by the (open or closed) disks of radius r centered at the points in $\mathcal{P}_n(\Omega)$ (respectively, $\mathcal{X}_n(\Omega)$). Let $K_{s,n}$ (respectively, $K'_{s,n}$) denote the event that $\sqrt{s}\Omega$ is $(k+1)$-covered by the unit-area (closed or open) disks centered at the points in $\mathcal{P}_n(\sqrt{s}\Omega)$ (respectively, $\mathcal{X}_n(\sqrt{s}\Omega)$). To simplify the presentation, let η denote the peripheral of Ω, which is equal to 4 (respectively, $2\sqrt{\pi}$) if Ω is a square (respectively, disk). For any $\xi \in \mathbb{R}$, let

$$\alpha(\xi) = \begin{cases} \frac{\left(\frac{\sqrt{\pi}\eta}{2}+e^{-\frac{\xi}{2}}\right)^2}{16\left(2\sqrt{\pi}\eta+e^{-\frac{\xi}{2}}\right)} e^{-\frac{\xi}{2}}, & \text{if } k=0; \\ \frac{\sqrt{\pi}\eta}{2^{k+6}(k+2)!} e^{-\frac{\xi}{2}}, & \text{if } k \ge 1. \end{cases}$$

and

$$\beta(\xi) = \begin{cases} 4e^{-\xi} + 2\left(\sqrt{\pi}+\frac{1}{\sqrt{\pi}}\right)\eta e^{-\frac{\xi}{2}}, & \text{if } k=0; \\ \frac{\sqrt{\pi}+\frac{1}{\sqrt{\pi}}}{2^{k-1}k!}\eta e^{-\frac{\xi}{2}}, & \text{if } k \ge 1. \end{cases}$$

The asymptotics of $\Pr\left[C_{n,r}\right]$ and $\Pr\left[C'_{n,r}\right]$ as n approaches infinity, and the asymptotics of $\Pr\left[K_{s,n}\right]$ and $\Pr\left[K'_{s,n}\right]$ as s approaches infinity are given in the following two theorems [4,10,16].

Theorem 3 *Let* $r_n = \sqrt{\frac{\ln n+(2k+1)\ln\ln n+\xi_n}{\pi n}}$. *If* $\lim_{n\to\infty}\xi_n = \xi$ *for some* $\xi \in \mathbb{R}$, *then*

$$1-\beta(\xi) \le \lim_{n\to\infty}\Pr\left[C_{n,r_n}\right] \le \frac{1}{1+\alpha(\xi)} , \textit{ and}$$
$$1-\beta(\xi) \le \lim_{n\to\infty}\Pr\left[C'_{n,r_n}\right] \le \frac{1}{1+\alpha(\xi)} .$$

If $\lim_{n\to\infty}\xi_n = \infty$, *then*

$$\lim_{n\to\infty}\Pr\left[C_{n,r_n}\right] = \lim_{n\to\infty}\Pr\left[C'_{n,r_n}\right] = 1 .$$

If $\lim_{n\to\infty}\xi_n = -\infty$,*then*

$$\lim_{n\to\infty}\Pr\left[C_{n,r_n}\right] = \lim_{n\to\infty}\Pr\left[C'_{n,r_n}\right] = 0 .$$

Theorem 4 *Let* $\mu(s) = \ln s + 2(k+1)\ln\ln s + \xi(s)$. *If* $\lim_{s\to\infty}\xi(s) = \xi$ *for some* $\xi \in \mathbb{R}$, *then*

$$1-\beta(\xi) \le \lim_{s\to\infty}\Pr\left[K_{s,\mu(s)s}\right] \le \frac{1}{1+\alpha(\xi)} , \textit{ and}$$
$$1-\beta(\xi) \le \lim_{s\to\infty}\Pr\left[K'_{s,\mu(s)s}\right] \le \frac{1}{1+\alpha(\xi)} .$$

If $\lim_{s\to\infty}\xi(s) = \infty$, *then*

$$\lim_{s\to\infty}\Pr\left[K_{s,\mu(s)s}\right] = \lim_{s\to\infty}\Pr\left[K'_{s,\mu(s)s}\right] = 1 .$$

If $\lim_{s\to\infty}\xi(s) = -\infty$, *then*

$$\lim_{s\to\infty}\Pr\left[K_{s,\mu(s)s}\right] = \lim_{s\to\infty}\Pr\left[K'_{s,\mu(s)s}\right] = 0 .$$

In Gabriel graphs (GG), two nodes have an edge if and only if there is no other node in the disk using the segment of these two nodes as its diameter. If V is a point set and l is a positive real number, we use $\rho_{GG}(V)$ to denote the largest edge length of the GG over V, and $N(V,l)$ denotes the number of GG edges over V whose length is at least l. Wan and Yi (2007) [11] gave the following theorem.

Theorem 5 *Let* Ω *be a unit-area disk. For any constant* ξ, $N\left(\mathcal{P}_n(\Omega), 2\sqrt{\frac{\ln n+\xi}{\pi n}}\right)$ *is asymptotically Poisson with mean* $2e^{-\xi}$, *and*

$$\lim_{n\to\infty}\Pr\left[\rho_{GG}\left(\mathcal{P}_n(\Omega)\right) < 2\sqrt{\frac{\ln n+\xi}{\pi n}}\right] = \exp\left(-2e^{-\xi}\right) .$$

Let $\rho_{\mathrm{Del}}(V)$ denote the largest edge length of the Delaunay triangulation over a point set V. The following theorem is given by Kozma et al. [3].

Theorem 6 *Let* Ω *be a unit-area disk. Then,*

$$\rho_{Del}\left(\mathcal{X}_n(\Omega)\right) = O\left(\sqrt[3]{\frac{\ln n}{n}}\right) .$$

In wireless networks with greedy forward routing (GFR), each node discards a packet if none of its neighbors is closer to the destination of the packet than itself, or otherwise forwards the packet to the neighbor that is the closest to the destination. Since each node only needs to maintain the locations of its one-hop neighbors and each packet should contain the location of the destination node, GFR can be implemented in a localized and memoryless manner. Because of the existence of local minima where none of the neighbors is closer to the destination than the current node, a packet may be discarded before it reaches its destination. To ensure that every packet can reach its destination, all nodes should have sufficiently large transmission radii to avoid the existence of local minima. Applying the r-disk model, we assume every node has the same transmission radius r, and each pair of nodes with distance at most r has a link. For a point set V, the *critical transmission radius* for GFR is given by

$$\rho_{\mathrm{GFR}}(V) = \max_{(u,v)\in V^2, u\neq v} \left(\min_{\|w-v\|<\|u-v\|} \|w-u\| \right) .$$

In the definition, (u,v) is a source–destination pair and w is a node that is closer to v than u. If every node is with a transmission radius not less than $\rho_{\mathrm{GFR}}(V)$, GFR can guarantee the deliverability between any source–destination pair [12].

Theorem 7 *Let Ω be a unit-area convex compact region with bounded curvature, and $\beta_0 = 1/\left(2/3 - \sqrt{3}/2\pi\right) \approx 1.6^2$. Suppose that $n\pi r_n^2 = (\beta + o(1)) \ln n$ for some $\beta > 0$. Then,*

1. *If $\beta > \beta_0$, then $\rho_{GFR}(\mathcal{P}_n(\Omega)) \le r_n$ is asymptotically almost sure.*
2. *If $\beta < \beta_0$, then $\rho_{GFR}(\mathcal{P}_n(\Omega)) > r_n$ is asymptotically almost sure.*

Applications

In the literature, r-disk graphs (or unit disk graphs by proper scaling) are widely used to model homogeneous wireless networks in which each node is equipped with an omnidirectional antenna. According to the path loss of radio frequency, the transmission ranges (radii) of wireless devices depend on transmission powers. For simplicity, the power assignment problem usually is modeled by a corresponding transmission range assignment problem. Recently, wireless ad-hoc networks have attracted attention from a lot of researchers because of various possible applications. In many of the possible applications, since wireless devices are powered by batteries, transmission range assignment has become one of the most important tools for prolonging system lifetime. By applying the theory of critical ranges, a randomly deployed wireless ad-hoc network may have good properties in high probability if the transmission range is larger than some critical value.

One application of critical ranges is to connectivity of networks. A network is k-vertex-connected if there exist k node-disjoint paths between any pair of nodes. With such a property, at least k distinct communication paths exist between any pair of nodes, and the network is connected even if $k-1$ nodes fail. Thus, with a higher degree of connectivity, a network may have larger bandwidth and higher fault tolerance capacity. In addition, in [9,14], and [15], networks with node or link failures were considered.

Another application is in topology control. To efficiently operate wireless ad-hoc networks, subsets of network topology will be constructed and maintained. The related topics are called topology control. A spanner is a subset of the network topology in which the minimal total cost of a path between any pair of nodes, e. g. distance or energy consumption, is only a constant fact larger than the minimal total cost in the original network topology. Hence spanners are good candidates for virtual backbones. Geometric structures, including Euclidean minimal spanning trees, relative neighbor graphs, Gabriel graphs, Delaunay triangulations, Yao's graphs, etc., are widely used ingredients to construct spanners [1,5,13]. By applying the knowledge of critical ranges, the complexity of algorithm design can be reduced, e. g. [3,11].

Open Problems

A number of problems related to critical ranges remain open. Most problems discussed here apply 2-D plane geometry. In other words, the point set is in the plane. The first direction for future work is to study those problems in high-dimension spaces. Another open research area is on the longest-edge problems for other geometric structures, e. g. relative neighbor graphs and Yao's graphs. A third direction for future work involves considering relations between graph properties. A well-known result in random geometric graphs is that vanishment of isolated nodes asymptotically implies connectivity of networks. But for the wireless networks with unreliable links, this property is still open. In addition, in wireless sensor networks, the relations between connectivity and coverage are also interesting.

Cross References

- ► Applications of Geometric Spanner Networks
- ► Connected Dominating Set
- ► Dilation of Geometric Networks
- ► Geometric Spanners

► Minimum Geometric Spanning Trees
► Minimum *k*-Connected Geometric Networks
► Randomized Broadcasting in Radio Networks
► Randomized Gossiping in Radio Networks

Recommended Reading

1. Cartigny, J., Ingelrest, F., Simplot-Ryl, D., Stojmenovic, I.: Localized LMST and RNG based minimum-energy broadcast protocols in ad hoc networks. Ad Hoc Netw. **3**(1), 1–16 (2004)
2. Dette, H., Henze, N.: The limit distribution of the largest nearest-neighbour link in the unit *d*-cube. J. Appl. Probab. **26**, 67–80 (1989)
3. Kozma, G., Lotker, Z., Sharir, M., Stupp, G.: Geometrically aware communication in random wireless networks. In: Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing, 25–28 July 2004, pp. 310–319
4. Kumar, S., Lai, T.H., Balogh, J.: On *k*-coverage in a mostly sleeping sensor network. In: Proceedings of the 10th Annual International Conference on Mobile Computing and Networking (MobiCom'04), 26 Sept–1 Oct 2004
5. Li, N., Hou, J.C., Sha, L.: Design and analysis of a MST-based distributed topology control algorithm for wireless ad-hoc networks. In: 22nd Annual Joint Conference Of The IEEE Computer And Communications Societies (INFOCOM 2003), vol. 3, 1–3 April 2003, pp. 1702–1712
6. Penrose, M.: The longest edge of the random minimal spanning tree. Ann. Appl. Probab. **7**(2), 340–361 (1997)
7. Penrose, M.: On *k*-connectivity for a geometric random graph. Random. Struct. Algorithms **15**(2), 145–164 (1999)
8. Penrose, M.: Random Geometric Graphs. Oxford University Press, Oxford (2003)
9. Wan, P.-J., Yi, C.-W.: Asymptotic critical transmission ranges for connectivity in wireless ad hoc networks with Bernoulli nodes. In: IEEE Wireless Communications and Networking Conference (WCNC 2005), 13–17 March 2005
10. Wan, P.-J., Yi, C.-W.: Coverage by randomly deployed wireless sensor networks. In: Proceedings of the 4th IEEE International Symposium on Network Computing and Applications (NCA 2005), 27–29 July 2005
11. Wan, P.-J., Yi, C.-W.: On the longest edge of Gabriel graphs in wireless ad hoc networks. Trans. Parallel Distrib. Syst. **18**(1), 1–16 (2007)
12. Wan, P.-J., Yi, C.-W., Yao, F., Jia, X.: Asymptotic critical transmission radius for greedy forward routing in wireless ad hoc networks. In: Proceedings of the 7th ACM International Symposium on Mobile Ad Hoc Networking and Computing, 22–25 May 2006, pp. 25–36
13. Wang, Y., Li, X.-Y.: Localized construction of bounded degree and planar spanner for wireless ad hoc networks, In: Proceedings of the 2003 joint workshop on Foundations of mobile computing (DIALM-POMC'03), 19 Sept 2003, pp. 59–68
14. Yi, C.-W., Wan, P.-J., Li, X.-Y., Frieder, O.: Asymptotic distribution of the number of isolated nodes in wireless ad hoc networks with Bernoulli nodes. In: IEEE Wireless Communications and Networking Conference (WCNC 2003), March 2003
15. Yi, C.-W., Wan, P.-J., Lin, K.-W., Huang, C.-H.: Asymptotic distribution of the Number of isolated nodes in wireless ad hoc networks with unreliable nodes and links. In: the 49th Annual IEEE GLOBECOM Technical Conference (GLOBECOM 2006), 27 Nov–1 Dec 2006
16. Zhang, H., Hou, J.: On deriving the upper bound of α-lifetime for large sensor networks. In: Proceedings of the 5th ACM International Symposium on Mobile Ad Hoc Networking & Computing (MobiHoc 2004), 24–26 March 2004

Cryptographic Hardness of Learning

1994; Kearns, Valiant

Adam Klivans
Department of Computer Science, University of Texas at Austin, Austin, TX, USA

Keywords and Synonyms

Representation-independent hardness for learning

Problem Definition

This paper deals with proving negative results for distribution-free PAC learning. The crux of the problem is proving that a polynomial-time algorithm for learning various concept classes in the PAC model implies that several well-known cryptosystems are insecure. Thus, if we assume a particular cryptosystem is secure we can conclude that it is impossible to efficiently learn a corresponding set of concept classes.

PAC Learning

We recall here the PAC learning model. Let C be a concept class (a set of functions over n variables), and let D be a distribution over the input space $\{0, 1\}^n$. With C we associate a size function size that measures the complexity of each $c \in C$. For example if C is a class of Boolean circuits then size(c) is equal to the number of gates in c. Let A be a randomized algorithm that has access to an oracle which returns labeled examples $(x, c(x))$ for some unknown $c \in C$; the examples x are drawn according to D. Algorithm A PAC-learns concept class C by hypothesis class H if for any $c \in C$, for any distribution D over the input space, and any $\epsilon, \delta > 0$, A runs in time $poly(n, 1/\epsilon, 1/\delta, size(c))$ and produces a hypothesis $h \in H$ such that with probability at least $(1 - \delta)$, $Pr_D[c(x) \neq h(x)] < \epsilon$. This probability is taken over the random coin tosses of A as well as over the random labeled examples seen from distribution D. When $H = C$ (the hypothesis must be some concept in C) then A is a *proper* PAC learning algorithm. In this article is not assumed $H = C$, i. e. hardness results for *representation-independent* learning algorithms are discussed. The only assumption made on H is that for each $h \in H$, h can be evaluated in polynomial time for every input of length n.

Cryptographic Primitives

Also required is knowledge of various cryptographic primitives such as public-key cryptosystems, one-way functions, and one-way trapdoor functions. For a formal treatment of these primitives refer to Goldreich [3].

Informally, a function f is *one-way* if, after choosing a random x of length n and giving an adversary A only $f(x)$, it is computationally intractable for A to find y such that $f(y) = f(x)$. Furthermore, given x, $f(x)$ can be evaluated in polynomial time. That is, f is easy to compute one-way, but there is no polynomial-time algorithm for finding pre-images of f on randomly chosen inputs. Say a function f is *trapdoor* if f is one-way, but if an adversary A is given access to a secret "trapdoor" d, then A *can* efficiently find random pre-images of f.

Trapdoor functions that are permutations are closely related to *public-key cryptosystems*: imagine a person Alice who wants to allow others to secretly communicate with her. She publishes a one-way trapdoor permutation f so that it is publicly available to everyone, but keeps the "trapdoor" d to herself. Then Bob can send Alice a secret message x by sending her $f(x)$. Only Alice is able to invert f (recall f is a permutation) and recover x because only she knows d.

Key Results

The main insight in Kearns and Valiant's work is the following: if f is a trapdoor one-way function, and C is a circuit class containing the set of functions capable of inverting f given access to the trap-door, then C is not efficiently PAC learnable. I. e., assuming the difficulty of inverting trap-door function f, there is a distribution on $\{0, 1\}^n$ where no learning algorithm can succeed in learning f's associated decryption function.

The following theorem is stated in the (closely related) language of public-key cryptosystems:

Theorem 1 (cryptography and learning; cf. Kearns & Valiant[4]) *Consider a public-key cryptosystem for encrypting individual bits into n-bit strings. Let C be a concept class that contains all the decryption functions* $\{0, 1\}^n \to \{0, 1\}$ *of the cryptosystem. If C is PAC-learnable in polynomial time then there is a polynomial-time distinguisher between the encryptions of 0 and 1.*

The intuition behind the proof is as follows: fix an encryption function f, associated secret key d, and let C be a class of functions such that the problem of inverting $f(x)$ given d can be computed by an element c of C; notice that knowledge of d is not necessary to generate a polynomial-size sample of $(x, f(x))$ pairs.

If C is PAC learnable, then given a relatively small number of encrypted messages $(x, f(x))$, a learning algorithm A can find a hypothesis h that will approximate c and thus have a non-negligible advantage for decrypting future *randomly* encrypted messages. This violates the security properties of the cryptosystem.

A natural question follows: "what is the simplest concept class that can compute the decryption function for secure public-key cryptosystems?" For example, if a public-key cryptosystem is proven to be secure, and encrypted messages can be decrypted (given the secret key) by polynomial-size DNF formulas, then, by Theorem 1, one could conclude that polynomial-size DNF formulas cannot be learned in the PAC model.

Kearns and Valiant do not obtain such a hardness result for learning DNF formulas (it is still an outstanding open problem), but they do obtain a variety of hardness results assuming the security of various well-known public-key cryptosystems based on the hardness of number-theoretic problems such as factoring.

The following list summarizes their main results:

- Let C be the class of polynomial-size Boolean formulas (not necessarily DNF formulas) or polynomial-size circuits of logarithmic depth. Assuming that the RSA cryptosystem is secure, or recognizing quadratic residues is intractable, or factoring Blum integers is intractable, C is not PAC learnable.
- Let C be the class of polynomial-size deterministic finite automata. Under the same assumptions as above, C is not PAC learnable.
- Let C be the class of constant depth threshold circuits of polynomial size. Under the same assumptions as above, C is not PAC learnable. The depth of the circuit class is not specified but it can be seen to be at most 4.

Kearns and Valiant also prove the intractability of finding optimal solutions to related coloring problems assuming the security of the above cryptographic primitives (breaking RSA, for example).

Relationship to Hardness Results for Proper Learning

The key results above should not be confused with the extensive literature regarding hardness results for *properly* PAC learning concept classes. For example, it is known [1] that, unless RP=NP, it is impossible to properly PAC learn polynomial-size DNF formulas (i. e., require the learner to learn DNF formulas by outputting a DNF formula as its hypothesis). Such results are *incomparable* to the work of Kearns and Valiant, as they require something much stronger from the learner but take a much weaker assumption (RP=NP is a weaker assumption than the assumption that RSA is secure).

Applications and Related Work

Valiant [10] was the first to observe that the existence of a particular cryptographic primitive (pseudorandom functions) implies hardness results for PAC learning concept classes. The work of Kearns and Valiant has subsequently found many applications. Klivans and Sherstov have recently shown [7] that the problem of PAC learning intersections of halfspaces (a very simple depth-2 threshold circuit) is intractable unless certain lattice-based cryptosystems due to Regev [9] are not secure. Their result makes use of the Kearns and Valiant approach. Angluin and Kharitonov [2] have extended the Kearns and Valiant paradigm to give cryptographic hardness results for learning concept classes even if the learner has *query access* to the unknown concept. Kharitonov [6] has given hardness results for learning polynomial-size, constant depth circuits that assumes the existence of secure pseudorandom generators rather than the existence of public-key cryptosystems.

Open Problems

The major open problem in this line of research is to prove a cryptographic hardness result for PAC learning polynomial-size DNF formulas. Currently, polynomial-size DNF formulas seem far too weak to compute cryptographic primitives such as the decryption function for a well-known cryptosystem. The fastest known algorithm for PAC learning DNF formulas runs in time $2^{\tilde{O}(n^{1/3})}$ [8].

Cross References

► PAC Learning

Recommended Reading

1. Alekhnovich, M., Braverman, M., Feldman, V., Klivans, A. R., Pitassi, T.: Learnability and automatizability. In: Proceedings of the 45th Symposium on Foundations of Computer Science, 2004
2. Angluin, D., Kharitonov, M.: When Won't Membership Queries Help? J. Comput. Syst. Sci. **50**, (1995)
3. Goldreich, O.: Foundations of Cryptography: Basic Tools. Cambridge University Press (2001)
4. Kearns, M., Valiant, L.: Cryptographic limitations on learning Boolean formulae and finite automata. J. ACM **41**(1), 67–95 (1994)
5. Kearns, M., Vazirani, U.: An introduction to computational learning theory. MIT Press, Cambridge (1994)
6. Kharitonov, M.: Cryptographic hardness of distribution-specific learning. In: Proceedings of the Twenty-Fifth Annual Symposium on Theory of Computing, 1993, pp. 372–381
7. Klivans, A. , Sherstov, A. A.: Cryptographic Hardness for Learning Intersections of Halfspaces. In: Proceedings of the 47th Symposium on Foundations of Computer Science, 2006
8. Klivans, A., Servedio, R.: Learning DNF in time $2^{\tilde{O}(n^{1/3})}$. In: Proceedings of the 33rd Annual Symposium on Theory of Computing, 2001
9. Regev, O.: New Lattice-Based Cryptographic Constructions. J. ACM **51**, 899–942 (2004)
10. Valiant, L.: A theory of the learnable. Commun. ACM **27**(11), 1134–1142 (1984)

Cuckoo Hashing

2001; Pagh, Rodler

RASMUS PAGH
Computational Logic and Algorithms Group,
IT University of Copenhagen, Copenhagen, Denmark

Problem Definition

A *dictionary* (also known as an *associative array*) is an abstract data structure capable of storing a set S of elements, referred to as *keys*, and information associated with each key. The operations supported by a dictionary are insertion of a key (and associated information), deletion of a key, and lookup of a key (retrieving the associated information). In case a lookup is made on a key that is not in S, this must be reported by the data structure.

Hash tables is a class of data structures used for implementing dictionaries in the RAM model of computation. *Open addressing* hash tables is a particularly simple type of hash tables, where the data structure is an array such that each entry either contains a key of S or is marked "empty". *Cuckoo hashing* addresses the problem of implementing an open addressing hash table with *worst case* constant lookup time. Specifically, a constant number of entries in the hash table should be associated with each key x, such that x is present in one of these entries if $x \in S$.

In the following it is assumed that a key as well as the information associated with a key are single machine words. This is essentially without loss of generality: If more associated data is wanted, it can be referred to using a pointer. If keys are longer than one machine word, they can be mapped down to a single (or a few) machine words using universal hashing [3], and the described method used on the hash values (which are unique to each key with high probability). The original key must then be stored as associated data. Let n denote an upper bound on the size of S. To allow the size of the set to grow beyond n, *global rebuilding* can be used.

Key Results

Prehistory

It has been known since the advent of universal hashing [3] that if the hash table has $r \geq n^2$ entries, a lookup can be implemented by retrieving just a single entry in the hash table. This is done by storing a key x in entry $h(x)$ of the hash table, where h is a function from the set of machine words to $\{1, \ldots, n^2\}$. If h is chosen at random from a *universal family* of hash functions [3] then with probability at least 1/2 every key in S is assigned a unique entry. The same behavior would be seen if h was a random function, but in contrast to random functions there are universal families that allow efficient storage and evaluation of h (constant number of machine words, and constant evaluation time).

This overview concentrates on the case where the space used by the hash table is linear, $r = O(n)$. It was shown by Azar et al. [1] that it is possible to combine linear space with worst case constant lookup time. It was not considered how to construct the data structure. Since randomization is used, all schemes discussed have a probability of error. However, this probability is small, $O(1/n)$ or less for all schemes, and an error can be handled by *rehashing* (changing the hash functions and rebuilding the hash table). The result of [1] was shown under the assumption that the algorithm is given free access to a number of truly random hash functions. In many of the subsequent papers it is shown how to achieve the bounds using explicitly defined hash functions. However, no attempt is made here to cover these constructions.

In the following, let ε denote an arbitrary positive constant. Pagh [9] showed that retrieving two entries from the hash table suffices when $r \geq (2+\varepsilon)n$. Specifically, lookup of a key x can be done by retrieving entries $h_1(x)$ and $h_2(x)$ of the hash table, where h_1 and h_2 are random hash functions mapping machine words to $\{1, \ldots, r\}$. The same result holds if h_1 has range $\{1, \ldots, r/2\}$ and h_2 has range $\{r/2+1, \ldots, r\}$, that is, if the two lookups are done in disjoint parts of memory.

It follows from [9] that it is not possible to perform lookup by retrieving a *single* entry in the worst case unless the hash table is of size $n^{2-o(1)}$.

Cuckoo Hashing

Pagh and Rodler [10] showed how to maintain the data structure of Pagh [9] under insertions. They considered the variant in which the lookups are done in disjoint parts of the hash table. It will be convenient to think of these as separate arrays, T_1 and T_2. Let $\perp$ denote the contents of an empty hash table entry, and let $x \leftrightarrow y$ express that the values of variables x and y are swapped. The proposed dynamic algorithm, called *cuckoo hashing*, performs insertions by the following procedure:

```
procedure insert($x$)
  $i := 1$;
  repeat
    $x \leftrightarrow T_i[h_i(x)]$; $i := 3 - i$;
  until $x = \perp$
end
```

At any time the variable x holds a key that needs to be inserted in the table, or $\perp$. The value of i changes between 1 and 2 in each iteration, so the algorithm is alternately exchanging the contents of x with a key from Table 1 and Table 2. Conceptually, what happens is that the algorithm moves a sequence of zero or more keys from one table to the other to make room for the new key. This is done in a greedy fashion, by kicking out any key that may be present in the location where a key is being moved. The similarity of the insertion procedure and the nesting habits of the European cuckoo is the reason for the name of the algorithm.

The pseudocode above is slightly simplified. In general the algorithm needs to make sure not to insert the same key twice, and handle the possibility that the insertion may not succeed (by rehashing if the loop takes too long).

Theorem 1 *Assuming that $r \geq (2+\varepsilon)n$ the expected time for the cuckoo hashing insertion procedure is $O(1)$.*

Generalizations of Cuckoo Hashing

Cuckoo hashing has been generalized in two directions. First of all, consider the case of k hash functions, for $k > 2$. Second, the hash table may be divided into "buckets" of size b, such that the lookup procedure searches an entire bucket for each hash function. Let (k, b)-cuckoo denote a scheme with k hash functions and buckets of size b. What was described above is a $(2, 1)$-*cuckoo* scheme. Already in 1999, $(4, 1)$-cuckoo was described in a patent application by David A. Brown (US patent 6,775,281). Fotakis et al. described and analyzed a $(k, 1)$-cuckoo scheme in [7], and a $(2, b)$-cuckoo scheme was described and analyzed by Dietzfelbinger and Weidling [4]. In both cases, it was shown that space utilization arbitrarily close to 100% is possible, and that the necessary fraction of unused space decreases exponentially with k and b. The insertion procedure considered in [4,7] is a breadth first search for the shortest sequence of key moves that can be made to accommodate the new key. Panigrahy [11] studied $(2, 2)$-cuckoo schemes in detail, showing that a space utilization of 83% can be achieved dynamically, still supporting constant time insertions using breadth first search. Independently, Fernholz and Ramachandran [6] and Cain, Sanders, and Wormald [2] determined the highest possible space utilization for $(2, k)$-cuckoo hashing in a *static* setting with no insertions. For $k = 2, 3, 4, 5$ the maximum space utilization is roughly 90%, 96%, 98%, and 99%, respectively.

Applications

Dictionaries have a wide range of uses in computer science and engineering. For example, dictionaries arise in many applications in string algorithms and data structures, database systems, data compression, and various information retrieval applications. No attempt is made to survey these further here.

Open Problems

The results above provide a good understanding of the properties of open addressing schemes with worst case constant lookup time. However, several aspects are still not understood satisfactorily.

First of all, there is no practical class of hash functions for which the above results can be shown. The only explicit classes of hash functions that are known to make the methods work either have evaluation time $\Theta(\log n)$ or use space $n^{\Omega(1)}$. It is an intriguing open problem to construct a class having constant evaluation time and space usage.

For the generalizations of cuckoo hashing the use of breadth first search is not so attractive in practice, due to the associated overhead in storage. A simpler approach that does not require any storage is to perform a random walk where keys are moved to a random, alternative position. (This generalizes the cuckoo hashing insertion procedure, where there is only one alternative position to choose.) Panigrahy [11] showed that this works for $(2, 2)$-cuckoo when the space utilization is low. However, it is unknown whether this approach works well as the space utilization approaches 100%.

Finally, many of the analyzes that have been given are not tight. In contrast, most classical open addressing schemes have been analyzed very precisely. It seems likely that precise analysis of cuckoo hashing and its generalizations is possible using techniques from analysis of algorithms, and tools from the theory of random graphs. In particular, the relationship between space utilization and insertion time is not well understood. A precise analysis of the probability that cuckoo hashing fails has been given by Kutzelnigg [8].

Experimental Results

All experiments on cuckoo hashing and its generalizations so far presented in the literature have been done using simple, heuristic hash functions. Pagh and Rodler [10] presented experiments showing that, for space utilization 1/3, cuckoo hashing is competitive with open addressing schemes that do not give a worst case guarantee. Zukowski et al. [12] showed how to implement cuckoo hashing such that it runs very efficiently on pipelined processors with the capability of processing several instructions in parallel. For hash tables that are small enough to fit in cache, cuckoo hashing was 2 to 4 times faster than chained hashing in their experiments. Erlingsson et al. [5] considered (k, b)-cuckoo schemes for various combinations of small values of k and b, showing that very high space utilization is possible even for modestly small values of k and b. For example, a space utilization of 99.9% is possible for $k = b = 4$. It was further found that the resulting algorithms were very robust. Experiments in [7] indicate that the random walk insertion procedure performs as well as one could hope for.

Cross References

► Dictionary Matching and Indexing (Exact and with Errors)
► Load Balancing

Recommended Reading

1. Azar, Y., Broder, A.Z., Karlin, A.R., Upfal, E.: Balanced allocations. SIAM J. Comput. **29**(1), 180–200 (1999)
2. Cain, J.A., Sanders, P., Wormald, N.: The random graph threshold for k-orientability and a fast algorithm for optimal multiple-choice allocation. In: Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '07), pp. 469–476. ACM Press, New Orleans, Louisiana, USA, 7–9 December 2007
3. Carter, J.L., Wegman, M.N.: Universal classes of hash functions. J. Comput. Syst. Sci. **18**(2), 143–154 (1979)
4. Dietzfelbinger, M., Weidling, C.: Balanced allocation and dictionaries with tightly packed constant size bins. In: ICALP. Lecture Notes in Computer Science, vol. 3580, pp. 166–178. Springer, Berlin (2005)
5. Erlingsson, Ú., Manasse, M., McSherry, F.: A cool and practical alternative to traditional hash tables. In: Proceedings of the 7th Workshop on Distributed Data and Structures (WDAS '06), Santa Clara, CA, USA, 4–6 January 2006
6. Fernholz, D., Ramachandran, V.: The k-orientability thresholds for $g_{n,p}$. In: Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '07), pp. 459–468. ACM Press, New Orleans, Louisiana, USA, 7–9 December 2007

7. Fotakis, D., Pagh, R., Sanders, P., Spirakis, P.G.: Space efficient hash tables with worst case constant access time. Theor. Comput. Syst. **38**(2), 229–248 (2005)
8. Kutzelnigg, R.: Bipartite Random Graphs and Cuckoo Hashing. In: Proc. Fourth Colloquium on Mathematics and Computer Science, Nancy, France, 18–22 September 2006
9. Pagh, R.: On the cell probe complexity of membership and perfect hashing. In: Proceedings of the 33rd Annual ACM Symposium on Theory of Computing (STOC '01), pp. 425–432. ACM Press, New York (2001)
10. Pagh, R., Rodler, F.F.: Cuckoo hashing. J. Algorithms **51**, 122–144 (2004)
11. Panigrahy, R.: Efficient hashing with lookups in two memory accesses. In: Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '05), pp. 830–839. SIAM, Vancouver, 23–25 January 2005
12. Zukowski, M., Heman, S., Boncz, P.A.: Architecture-conscious hashing. In: Proceedings of the International Workshop on Data Management on New Hardware (DaMoN), Article No. 6. ACM Press, Chicago, Illinois, USA, 25 June 2006

D

Data Migration

2004; Khuller, Kim, Wan

YOO-AH KIM
Computer Science and Engineering Department,
University of Connecticut, Storrs, CT, USA

Keywords and Synonyms

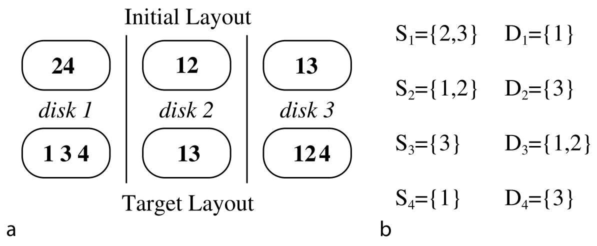
File transfers; Data movements

Problem Definition

The problem is motivated by the need to manage data on a set of storage devices to handle dynamically changing demand. To maximize utilization, the data layout (i. e., a mapping that specifies the subset of data items stored on each disk) needs to be computed based on disk capacities as well as the demand for data. Over time as the demand for data changes, the system needs to create new data layout. The data migration problem is to compute an efficient schedule for the set of disks to convert an initial layout to a target layout.

The problem is defined as follows. Suppose that there are N disks and Δ data items, and an initial layout and a target layout are given (see Fig. 1a for an example). For each item i, source disks S_i is defined to be a subset of disks which have item i in the initial layout. Destination disks D_i is a subset of disks that want to receive item i. In other words, disks in D_i have to store item i in the target layout but do not have to store it in the initial layout. Figure 1b shows the corresponding S_i and D_i. It is assumed that $S_i \neq \emptyset$ and $D_i \neq \emptyset$ for each item i. Data migration is the transfer of data to have all D_i receive data item i residing in S_i initially, and the goal is to minimize the total amount of time required for the transfers.

Assume that the underlying network is fully connected and the data items are all the same size. In other words, it takes the same amount of time to migrate an item from one disk to another. Therefore, migrations are performed



Data Migration, Figure 1
Left **An example of initial and target layout and** ***right*** **their corresponding** S_i**'s and** D_i**'s**

in rounds. Consider the half-duplex model, where each disk can participate in the transfer of only one item – either as a sender or as a receiver. The objective is to find a migration schedule using the minimum number of rounds. No bypass nodes[1] can be used and therefore all data items are sent only to disks that desire them.

Key Results

Khuller et al. [11] developed a 9.5-approximation for the data migration problem, which was later improved to $6.5 + o(1)$. In the next subsection, the lower bounds of the problem are first examined.

Notations and Lower Bounds

1. **Maximum in-degree (β):** Let β_j be the number of data items that a disk j has to receive. In other words, $\beta_j = |\{i|j \in D_i\}|$. Then $\beta = \max_j \beta_j$ is a lower bound on the optimal as a disk can receive only one data item in one round.
2. **Maximum number of items that a disk may be a source or destination for (α):** For each item i, at least one disk in S_i should be used as a source for the item, and this disk is called a *primary source*. A unique primary source $s_i \in S_i$ for each item i that minimizes

[1] A bypass node is a node that is not the target of a move operation, but is used as an intermediate holding point for a data item.

$\alpha = \max_{j=1,\ldots,N}(|\{i|j = s_i\}| + \beta_j)$ can be found using a network flow. Note that $\alpha \geq \beta$, and α is also a lower bound on the optimal solution.

3. **Minimum time required for cloning (M):** Let a disk j make a copy of item i at the kth round. At the end of the mth round, the number of copies that can be created from the copy is at most 2^{m-k} as in each round the number of copies can only be doubled. Also note that each disk can make a copy of only one item in one round. Since at least $|D_i|$ copies of item i need to be created, the minimum m that satisfies the following linear program gives a lower bound on the optimal solution: **L(m):**

$$\sum_j \sum_{k=1}^{m} 2^{m-k} x_{ijk} \geq |D_i| \quad \text{for all } i \tag{1}$$

$$\sum_i x_{ijk} \leq 1 \quad \text{for all } j, k \tag{2}$$

$$0 \leq x_{ijk} \leq 1 \tag{3}$$

Data Migration Algorithm

A 9.5-approximation can be obtained as follows. The algorithm first computes representative sets for each item and sends the item to the representative sets first, which in turn send the item to the remaining set. Representative sets are computed differently depending on the size of D_i.

Representatives for Big Sets For sets with size at least β, a *disjoint* collection of *representative sets* R_i, $i = 1 \ldots \Delta$ has to satisfy the following properties: Each R_i should be a subset of D_i and $|R_i| = \lfloor |D_i|/\beta \rfloor$. The representative sets can be found using a network flow.

Representatives for Small Sets For each item i, let $\gamma_i = |D_i| \bmod k$. A *secondary representative* r_i in D_i for the items with $\gamma_i \neq 0$ needs to be computed. A disk j can be a secondary representative r_i for several items as long as $\sum_{i \in I_j} \gamma_i \leq 2\beta - 1$, where I_j is a set of items for which j is a secondary representative. This can be done by applying the Shmoys–Tardos algorithm [17] for the generalized assignment problem.

Scheduling Migrations Given representatives for all data items, migrations can be done in three steps as follows:

1. **Migration to R_i:** Each item i is first sent to the set R_i. By converting a fractional solution given in $L(M)$, one can find a migration schedule from s_i to R_i and it requires at most $2M + \alpha$ rounds.
2. **Migration to r_i:** Item i is sent from primary source s_i to r_i. The migrations can be done in 1.5α rounds, using an algorithm for edge coloring [16].
3. **Migration to the remaining disks:** A transfer graph from representatives to the remaining disks can now be created as follows. For each item i, add directed edges from disks in R_i to $(\beta - 1)\lfloor \frac{|D_i|}{\beta} \rfloor$ disks in $D_i \setminus R_i$ such that the out-degree of each node in R_i is at most $\beta - 1$ and the in-degree of each node in $D_i \setminus R_i$ from R_i is 1. A directed edge is also added from the secondary representative r_i of item i to the remaining disks in D_i which do not have an edge coming from R_i. It has been shown that the maximum degree of the transfer graph is at most $4\beta - 5$ and the multiplicity is $\beta + 2$. Therefore, migration for the transfer graph can be done in $5\beta - 3$ rounds using an algorithm for multigraph edge coloring [18].

Analysis Note that the total number of rounds required in the algorithm described in "Data Migration Algorithm" is at most $2M + 2.5\alpha + 5\beta - 3$. As α, β and M are lower bounds on the optimal number of rounds, the abovementioned algorithm gives a 9.5-approximation.

Theorem 1 ([11]) *There is a 9.5-approximation algorithm for the data migration problem.*

Khuller et al. [10] later improved the algorithm and obtained a $(6.5 + o(1))$-approximation.

Theorem 2 ([10]) *There is a $(6.5 + o(1))$-approximation algorithm for the data migration problem.*

Applications

Data Migration in Storage Systems

Typically, a large storage server consists of several disks connected using a dedicated network, called a *storage area network*. To handle high demand, especially for multimedia data, a common approach is to replicate data objects within the storage system. Disks typically have constraints on storage as well as the number of clients that can access data from a single disk simultaneously. Approximation algorithms have been developed to map known demand for data to a specific data layout pattern to maximize utilization[2] [4,8,14,15]. In the layout, they compute not only how many copies of each item need to be created, but also a layout pattern that specifies the precise subset of items on each disk. The problem is NP-hard, but there are polynomial-time approximation schemes [4,8,14]. Given

[2]The utilization is the total number of clients that can be assigned to a disk that contains the data they want.

the relative demand for data, the algorithm computes an almost optimal layout.

Over time as the demand for data changes, the system needs to create new data layouts. To handle high demand for popular objects, new copies may have to be dynamically created and stored on different disks. The data migration problem is to compute a specific schedule for the set of disks to convert an initial layout to a target layout. Migration should be done as quickly as possible since the performance of the system will be suboptimal during migration.

Gossiping and Broadcasting

The data migration problem can be considered as a generalization of gossiping and broadcasting. The problems of gossiping and broadcasting play an important role in the design of communication protocols in various kinds of networks and have been extensively studied (see for example [6,7] and the references therein). The gossip problem is defined as follows. There are n individuals and each individual has an item of gossip that he/she wish to communicate to everyone else. Communication is typically done in rounds, where in each round an individual may communicate with at most one other individual. Some communication models allow for the full exchange of all items of gossip known to each individual in a single round. In addition, there may be a communication graph whose edge indicates which pairs of individuals are allowed to communicate directly in each round. In the broadcast problem, one individual needs to convey an item of gossip to every other individual. The data migration problem generalizes the gossiping and broadcasting in three ways: (1) each item of gossip needs to be communicated to only a subset of individuals; (2) several items of gossip may be known to an individual; (3) a single item of gossip can initially be shared by several individuals.

Open Problems

The data migration problem is NP-hard by reduction from the edge coloring problem. However, no inapproximability results are known for the problem. As the current best approximation factor is relatively high ($6.5 + o(1)$), it is an interesting open problem to narrow the gap between the approximation guarantee and the inapproximability.

Another open problem is to combine data placement and migration problems. This question was studied by Khuller et al. [9]. Given the initial layout and the new demand pattern, their goal was to find a set of data migrations that can be performed in a specific number of rounds and gives the best possible layout to the current demand pattern. They showed that even one-round migration is NP-hard and presented a heuristic algorithm for the one-round migration problem. The experiments showed that performing a few rounds of one-round migration consecutively works well in practice. Obtaining nontrivial approximation algorithms for this problem would be interesting future work.

Data migration in a heterogeneous storage system is another interesting direction for future research. Most research on data migration has focused mainly on homogeneous storage systems, assuming that disks have the same fixed capabilities and the network connections are of the same fixed bandwidth. In practice, however, large-scale storage systems may be heterogenous. For instance, disks tend to have heterogeneous capabilities as they are added over time owing to increasing demand for storage capacity. Lu et al. [13] studied the case when disks have variable bandwidth owing to the loads on different disks. They used a control-theoretic approach to generate adaptive rates of data migrations which minimize the degradation of the quality of the service. The algorithm reduces the latency experienced by clients significantly compared with the previous schemes. However, no theoretical bounds on the efficiency of data migrations were provided. Coffman et al. [2] studied the case when each disk i can handle p_i transfers simultaneously and provided approximation algorithms. Some papers [2,12] considered the case when the lengths of data items are heterogenous (but the system is homogeneous), and present approximation algorithms for the problem.

Experimental Results

Golubchik et al. [3] conducted an extensive study of the performance of data migration algorithms under different changes in user-access patterns. They compared the 9.5-approximation [11] and several other heuristic algorithms. Some of these heuristic algorithms cannot provide constant approximation guarantees, while for some of the algorithms no approximation guarantees are known. Although the worst-case performance of the algorithm by Khuller et al. [11] is 9.5, in the experiments the number of rounds required was less than 3.25 times the lower bound.

They also introduced the *correspondence problem*, in which a matching between disks in the initial layout with disks in the target layout is computed so as to minimize changes. A good solution to the correspondence problem can improve the performance of the data migration algorithms by a factor of 4.4 in their experiments, relative to a bad solution.

URL to Code

http://www.cs.umd.edu/projects/smart/data-migration/

Cross References

► Broadcasting in Geometric Radio Networks
► Deterministic Broadcasting in Radio Networks

Recommended Reading

A special case of the data migration problem was studied by Anderson et al. [1] and Hall et al. [5]. They assumed that a data transfer graph is given, in which a node corresponds to each disk and a directed edge corresponds to each move operation that is specified (the creation of new copies of data items is not allowed). Computing a data movement schedule is exactly the problem of edge-coloring the transfer graph. Algorithms for edge-coloring multigraphs can now be applied to produce a migration schedule since each color class represents a matching in the graph that can be scheduled simultaneously. Computing a solution with the minimum number of rounds is NP-hard, but several good approximation algorithms are available for edge coloring. With space constraints on the disk, the problem becomes more challenging. Hall et al. [5] showed that with the assumption that each disk has one spare unit of storage, very good constant factor approximations can be developed. The algorithms use at most $4\lceil \Delta/4 \rceil$ colors with at most $n/3$ bypass nodes, or at most $6\lceil \Delta/4 \rceil$ colors without bypass nodes.

Most of the results on the data migration problem deal with the half-duplex model. Another interesting communication model is the full-duplex model where each disk can act as a sender and a receiver in each round for a single item. There is a $(4 + o(1))$-approximation algorithm for the full-duplex model [10].

1. Anderson, E., Hall, J., Hartline, J., Hobbes, M., Karlin, A., Saia, J., Swaminathan, R., Wilkes, J.: An experimental study of data migration algorithms. In: Workshop on Algorithm Engineering (2001)
2. Coffman, E., Garey, M., Jr., Johnson, D., Lapaugh, A.: Scheduling file transfers. SIAM J. Comput. **14**(3), 744–780 (1985)
3. Golubchik, L., Khuller, S., Kim, Y., Shargorodskaya, S., Wan., Y.: Data migration on parallel disks. In: 12th Annual European Symposium on Algorithms (ESA) (2004)
4. Golubchik, L., Khanna, S., Khuller, S., Thurimella, R., Zhu, A.: Approximation algorithms for data placement on parallel disks. In: Symposium on Discrete Algorithms, pp. 223–232. Society for Industrial and Applied Mathematics, Philadelphia (2000)
5. Hall, J., Hartline, J., Karlin, A., Saia, J., Wilkes, J.: On algorithms for efficient data migration. In: SODA, pp. 620–629. Society for Industrial and Applied Mathematics, Philadelphia (2001)
6. Hedetniemi, S.M., Hedetniemi, S.T., Liestman, A.: A survey of gossiping and broadcasting in communication networks. Networks **18**, 129–134 (1988)
7. Hromkovic, J., Klasing, R., Monien, B., Peine, R.: Dissemination of information in interconnection networks (broadcasting and gossiping). In: Du, D.Z., Hsu, F. (eds.) Combinatorial Network Theory, pp. 125–212. Kluwer Academic Publishers, Dordrecht (1996)
8. Kashyap, S., Khuller, S.: Algorithms for non-uniform size data placement on parallel disks. In: Conference on FST&TCS Conference. LNCS, vol. 2914, pp. 265–276. Springer, Heidelberg (2003)
9. Kashyap, S., Khuller, S., Wan, Y-C., Golubchik, L.: Fast reconfiguration of data placement in parallel disks. In: Workshop on Algorithm Engineering and Experiments (2006)
10. Khuller, S., Kim, Y., Malekian, A.: Improved algorithms for data migration. In: 9th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (2006)
11. Khuller, S., Kim, Y., Wan, Y.-C.: Algorithms for data migration with cloning. SIAM J. Comput. **33**(2), 448–461 (2004)
12. Yoo-Ah Kim. Data migration to minimize the average completion time. J. Algorithms **55**, 42–57 (2005)
13. Lu, C., Alvarez, G.A., Wilkes, J.: Aqueduct:online data migration with performance guarantees. In: Proceedings of the Conference on File and Storage Technologies (2002)
14. Shachnai, H., Tamir, T.: Polynomial time approximation schemes for class-constrained packing problems. J. Sched. **4**(6) 313–338 (2001)
15. Shachnai, H., Tamir, T.: On two class-constrained versions of the multiple knapsack problem. Algorithmica **29**(3), 442–467 (2001)
16. Shannon, C.E.: A theorem on colouring lines of a network. J. Math. Phys. **28**, 148–151 (1949)
17. Shmoys, D.B., Tardos, E.: An approximation algorithm for the generalized assignment problem. Math. Program. **62**(3), 461–474 (1993)
18. Vizing, V.G.: On an estimate of the chromatic class of a p-graph (Russian). Diskret. Analiz. **3**, 25–30 (1964)

Data Reduction for Domination in Graphs

2004; Alber, Fellows, Niedermeier

ROLF NIEDERMEIER
Department of Math and Computer Science,
University of Jena, Jena, Germany

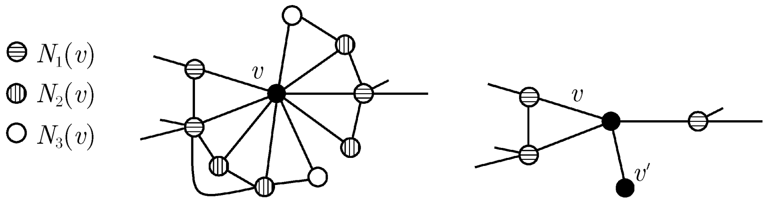
Keywords and Synonyms

Dominating set; Reduction to a problem kernel; Kernelization

Problem Definition

The NP-complete DOMINATING SET problem is a notoriously hard problem:

Problem 1 (Dominating Set)
INPUT: *An undirected graph $G = (V, E)$ and an integer $k \geq 0$.*



Data Reduction for Domination in Graphs, Figure 1
The *left-hand side* shows the partitioning of the neighborhood of a single vertex *v*. The *right-hand side* shows the result of applying the presented data reduction rule to this particular (sub)graph

QUESTION: *Is there an $S \subseteq V$ with $|S| \leq k$ such that every vertex $v \in V$ is contained in S or has at least one neighbor in S?*

For instance, for an n-vertex graph its optimization version is known to be polynomial-time approximable only up to a factor of $\Theta(\log n)$ unless some standard complexity-theoretic assumptions fail [9]. In terms of parametrized complexity, the problem is shown to be W[2]-complete [8]. Although still NP-complete when restricted to planar graphs, the situation much improves here. In her seminal work, Baker showed that there is an efficient polynomial-time approximation scheme (PTAS) [6], and the problem also becomes fixed-parameter tractable [2,4] when restricted to planar graphs. In particular, the problem becomes accessible to fairly effective data reduction rules and a kernelization result (see [16] for a general description of data reduction and kernelization) can be proven. This is the subject of this entry.

Key Results

The key idea behind the data reduction is preprocessing based on locally acting simplification rules. Exemplary, here we describe a rule where the local neighborhood of each graph vertex is considered. To this end, we need the following definitions.

We partition the neighborhood $N(v)$ of an arbitrary vertex $v \in V$ in the input graph into three disjoint sets $N_1(v)$, $N_2(v)$, and $N_3(v)$ depending on local neighborhood structure. More specifically, we define

- $N_1(v)$ to contain all neighbors of v that have edges to vertices that are not neighbors of v;
- $N_2(v)$ to contain all vertices from $N(v) \setminus N_1(v)$ that have edges to at least one vertex from $N_1(v)$;
- $N_3(v)$ to contain all neighbors of v that are neither in $N_1(v)$ nor in $N_2(v)$.

An example which illustrates such a partitioning is given in Fig. 1 (left-hand side). A helpful and intuitive interpretation of the partition is to see vertices in $N_1(v)$ as *exits* because they have direct connections to the world outside the closed neighborhood of v, vertices in $N_2(v)$ as *guards* because they have direct connections to exits, and vertices in $N_3(v)$ as *prisoners* because they do not see the world outside $\{v\} \cup N(v)$.

Now consider a vertex $w \in N_3(v)$. Such a vertex only has neighbors in $\{v\} \cup N_2(v) \cup N_3(v)$. Hence, to dominate w, at least one vertex of $\{v\} \cup N_2(v) \cup N_3(v)$ *must* be contained in a dominating set for the input graph. Since v can dominate all vertices that would be dominated by choosing a vertex from $N_2(v) \cup N_3(v)$ into the dominating set, we obtain the following data reduction rule.

If $N_3(v) \neq \emptyset$ for some vertex v, then remove $N_2(v)$
and $N_3(v)$ from G
and add a new vertex v'
with the edge $\{v, v'\}$ to G.

Note that the new vertex v' can be considered as a "gadget vertex" that "enforces" v to be chosen into the dominating set. It is not hard to verify the correctness of this rule, that is, the original graph has a dominating set of size k iff the reduced graph has a dominating set of size k. Clearly, the data reduction can be executed in polynomial time [5]. Note, however, that there are particular "diamond" structures that are not amenable to this reduction rule. Hence, a second, somewhat more complicated rule based on considering the joint neighborhood of *two* vertices has been introduced [5].

Altogether, the following core result could be shown [5].

Theorem 1 *A planar graph $G = (V, E)$ can be reduced in polynomial time to a planar graph $G' = (V', E')$ such that G has a dominating set of size k iff G' has a dominating set of size k and $|V'| = O(k)$.*

In other words, the theorem states that the DOMINATING SET in planar graphs has a linear-size problem kernel. The upper bound on $|V'|$ was first shown to be $335k$ [5] and was then further improved to $67k$ [7]. Moreover, the results can be extended to graphs of bounded genus [10]. In addition, similar results (linear kernelization) have been recently obtained for the FULL-DEGREE SPANNING TREE problem in planar graphs [13]. Very recently, these results have been generalized into a methodological framework [12].

Applications

DOMINATING SET is considered to be one of the most central graph problems [14,15]. Its applications range from facility location to bioinformatics.

Open Problems

The best lower bound for the size of a problem kernel for DOMINATING SET in planar graphs is $2k$ [7]. Thus, there is quite a gap between known upper and lower bounds. In addition, there have been some considerations concerning a generalization of the above-discussed data reduction rules [3]. To what extent such extensions are of practical use remains to be explored. Finally, a study of deeper connections between Baker's PTAS results [6] and linear kernelization results for DOMINATING SET in planar graphs seems to be worthwhile for future research. Links concerning the class of problems amenable to both approaches have been detected recently [12]. The research field of data reduction and problem kernelization as a whole together with its challenges is discussed in a recent survey [11].

Experimental Results

The above-described theoretical work has been accompanied by experimental investigations on synthetic as well as real-world data [1]. The results have been encouraging in general. However, note that grid structures seem to be a hard case where the data reduction rules remained largely ineffective.

Cross References

► Connected Dominating Set

Recommended Reading

1. Alber, J., Betzler, N., Niedermeier, R.: Experiments on data reduction for optimal domination in networks. Ann. Oper. Res. **146**(1), 105–117 (2006)
2. Alber, J., Bodlaender, H.L., Fernau, H., Kloks, T., Niedermeier, R.: Fixed parameter algorithms for Dominating Set and related problems on planar graphs. Algorithmica **33**(4), 461–493 (2002)
3. Alber, J., Dorn, B., Niedermeier, R.: A general data reduction scheme for domination in graphs. In: Proc. 32nd SOFSEM. LNCS, vol. 3831, pp. 137–147. Springer, Berlin (2006)
4. Alber, J., Fan, H., Fellows, M.R., Fernau, H., Niedermeier, R., Rosamond, F., Stege, U.: A refined search tree technique for Dominating Set on planar graphs. J. Comput. Syst. Sci. **71**(4), 385–405 (2005)
5. Alber, J., Fellows, M.R., Niedermeier, R.: Polynomial time data reduction for Dominating Set. J. ACM **51**(3), 363–384 (2004)
6. Baker, B.S.: Approximation algorithms for NP-complete problems on planar graphs. J. ACM **41**(1), 153–180 (1994)
7. Chen, J., Fernau, H., Kanj, I.A., Xia, G.: Parametric duality and kernelization: lower bounds and upper bounds on kernel size. SIAM J. Comput. **37**(4), 1077–1106 (2007)
8. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, New York (1999)
9. Feige, U.: A threshold of $\ln n$ for approximating set cover. J. ACM **45**(4), 634–652 (1998)
10. Fomin, F.V., Thilikos, D.M.: Fast parameterized algorithms for graphs on surfaces: Linear kernel and exponential speed-up. In: Proc. 31st ICALP. LNCS, vol. 3142, pp. 581–592. Springer, Berlin (2004)
11. Guo, J., Niedermeier, R.: Invitation to data reduction and problem kernelization. ACM SIGACT News **38**(1), 31–45 (2007)
12. Guo, J., Niedermeier, R.: Linear problem kernels for NP-hard problems on planar graphs. In: Proc. 34th ICALP. LNCS, vol. 4596, pp. 375–386. Springer, Berlin (2007)
13. Guo, J., Niedermeier, R., Wernicke, S.: Fixed-parameter tractability results for full-degree spanning tree and its dual. In: Proc. 2nd IWPEC. LNCS, vol. 4196, pp. 203–214. Springer, Berlin (2006)
14. Haynes, T.W., Hedetniemi, S.T., Slater, P.J.: Domination in Graphs: Advanced Topics. Pure and Applied Mathematics, vol. 209. Marcel Dekker, New York (1998)
15. Haynes, T.W., Hedetniemi, S.T., Slater, P.J.: Fundamentals of Domination in Graphs. Pure and Applied Mathematics, vol. 208. Marcel Dekker, New York (1998)
16. Niedermeier, R.: Invitation to Fixed-Parameter Algorithms. Oxford University Press, New York (2006)

Decoding

► Decoding Reed–Solomon Codes
► List Decoding near Capacity: Folded RS Codes

Decoding Reed–Solomon Codes

1999; Guruswami, Sudan

VENKATESAN GURUSWAMI
Department of Computer Science and Engineering,
University of Washington, Seattle, WA, USA

Keywords and Synonyms

Decoding; Error correction

Problem Definition

In order to ensure the integrity of data in the presence of errors, an *error-correcting code* is used to *encode* data into a redundant form (called a *codeword*). It is natural to view both the original data (or *message*) as well as the associated codeword as strings over a finite alphabet. Therefore, an error-correcting code C is defined by an injective encoding map $E : \Sigma^k \to \Sigma^n$, where k is called the *message length*, and n the *block length*. The codeword, being a redundant form of the message, will be longer than the message. The *rate* of an error-correcting code is defined as the ratio k/n of the length of the message to the length of the codeword. The rate is a quantity in the interval $(0, 1]$, and is a measure of the redundancy introduced by the code. Let $R(C)$ denote the rate of a code C.

The redundancy built into a codeword enables detection and hopefully also correction of any errors introduced, since only a small fraction of all possible strings will be legitimate codewords. Ideally, the codewords encoding different messages should be "far-off" from each other, so that one can recover the original codeword even when it is distorted by moderate levels of noise. A natural measure of distance between strings is the Hamming distance. The Hamming distance between strings $x, y \in \Sigma^*$ of the same length, denoted $\mathsf{dist}(x, y)$, is defined to be the number of positions i for which $x_i \neq y_i$.

The *minimum distance*, or simply *distance*, of an error-correcting code C, denoted $d(C)$, is defined to be the smallest Hamming distance between the encodings of two distinct messages. The *relative distance* of a code C of block length n, denoted $\delta(C)$, is the ratio between its distance and n. Note that arbitrary corruption of any $\lfloor (d(C) - 1)/2 \rfloor$ of locations of a codeword of C cannot take it closer (in Hamming distance) to any other codeword of C. Thus in principle (i. e., efficiency considerations apart) error patterns of at most $\lfloor (d(C) - 1)/2 \rfloor$ errors can be corrected. This task is called *unique decoding* or *decoding up to half-the-distance*. Of course, it is also possible, and will often be the case, that error patterns with more than $d(C)/2$ errors can also be corrected by decoding the string to the closest codeword in Hamming distance. The latter task is called *Nearest-Codeword decoding* or *Maximum Likelihood Decoding (MLD)*.

One of the fundamental trade-offs in the theory of error-correcting codes, and in fact one could say all of combinatorics, is the one between rate $R(C)$ and distance $d(C)$ of a code. Naturally, as one increases the rate and thus number of codewords in a code, some two codewords must come closer together thereby lowering the distance. More qualitatively, this represents the tension between the redundancy of a code and its error-resilience. To correct more errors requires greater redundancy, and thus lower rate.

A code defined by encoding map $E : \Sigma^k \to \Sigma^n$ with minimum distance d is said to be an (n, k, d) code. Since there are $|\Sigma|^k$ codewords and only $|\Sigma^{k-1}|$ possible projections onto the first $k = 1$ coordinates, some two codewords must agree on the first $k - 1$ positions, implying that the distance d of the code must obey $d \leq n - k + 1$ (this is called the *Singleton bound*). Quite surprisingly, over large alphabets Σ there are well-known codes called Reed–Solomon codes which meet this bound exactly and have the optimal distance $d = n - k + 1$ for any given rate k/n. (In contrast, for small alphabets, such as $\Sigma = \{0, 1\}$, the optimal trade-off between rate and relative distance for an asymptotic family of codes is unknown and is a major open question in combinatorics.)

This article will describe the best known algorithmic results for error-correction of Reed–Solomon codes. These are of central theoretical and practical interest given the above-mentioned optimal trade-off achieved by Reed–Solomon codes, and their ubiquitous use in our everyday lives ranging from compact disc players to deep-space communication.

Reed–Solomon Codes

Definition 1 A *Reed–Solomon code* (or RS code), $\mathsf{RS}_{\mathbb{F},S}[n, k]$, is parametrized by integers n, k satisfying $1 \leq k \leq n$, a finite field $\mathbb{F}$ of size at least n, and a tuple $S = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ of n *distinct* elements from $\mathbb{F}$. The code is described as a subset of $\mathbb{F}^n$ as:

$$\mathsf{RS}_{\mathbb{F},S}[n, k] = \{(p(\alpha_1), p(\alpha_2), \ldots, p(\alpha_n)) | p(X) \in \mathbb{F}[X] \text{ is a polynomial of degree} \leq k - 1\} .$$

In other words, the message is viewed as a polynomial, and it is encoded by evaluating the polynomial at n distinct field elements $\alpha_1, \ldots, \alpha_n$. The resulting code is linear of dimension k, and its minimum distance equals $n - k + 1$, which matches the Singleton bound.

The distance property of RS codes follows from the fact that the evaluations of two distinct polynomials of degree less than k can agree on at most $k - 1$ field elements. Note that in the absence of errors, given a codeword $\mathbf{y} \in \mathbb{F}^n$, one can recover its corresponding message by polynomial interpolation on any k out of the n codeword positions. In fact, this also gives an *erasure decoding* algorithm when all but the information-theoretically bare minimum necessary k symbols are erased from the codeword (but the receiver *knows* which symbols have been erased and the correct values of the rest of the symbols). The RS decoding problem, therefore, amounts to a noisy polynomial interpolation problem when some of the evaluation values are incorrect.

The holy grail in decoding RS codes would be to find the polynomial $p(X)$ whose RS encoding is closest in Hamming distance to a noisy string $\mathbf{y} \in \mathbb{F}^n$. One could then decode $\mathbf{y}$ to this message $p(X)$ as the maximum likelihood choice. No efficient algorithm for such nearest-codeword decoding is known for RS codes (or for that matter any family of "good" or non-trivial codes), and it is believed that the problem is NP-hard. Guruswami and Vardy [6] proved the problem to NP-hard over exponentially large fields, but this is a weak negative result since normally one considers Reed–Solomon codes over fields of size at most $O(n)$.

Given the intractability of nearest-codeword decoding in its extreme generality, lot of attention has been devoted to the *bounded distance decoding* problem, where one assumes that the string $\mathbf{y} \in \mathbb{F}^n$ to be decoded has at most e errors, and the goal is to find the Reed–Solomon codeword(s) within Hamming distance e from $\mathbf{y}$.

When $e < (n-k)/2$, this corresponds to decoding up to half the distance. This is a classical problem for which a polynomial time algorithm was first given by Peterson [8]. (It is notable that this even before the notion of polynomial time was put forth as the metric of theoretical efficiency.) The focus of this article is on a *list decoding* algorithm for Reed–Solomon codes due to Guruswami and Sudan [5] that decode beyond half the minimum distance. The formal problem and the key results are stated next.

Key Results

In this section, the main result of focus concerning decoding Reed–Solomon codes is stated. Given the target of decoding errors beyond half-the-minimum distance, one needs to deal with inputs where there may be more than one codeword within the radius e specified in the bounded distance decoding problem. This is achieved by a relaxation of decoding called *list decoding* where the decoder outputs a list of all codewords (or the corresponding messages) within Hamming distance e from the received word. If one wishes, one can choose the closest codeword among the list as the "most likely" answer, but there are many applications of Reed–Solomon decoding, for example to decoding concatenated codes and several applications in complexity theory and cryptography, where having the entire list of codewords adds to the power of the decoding primitive. The main result of Guruswami and Sudan [5], building upon the work of Sudan [9], is the following:

Theorem 1 ([5]) *Let $C = \mathrm{RS}_{\mathbb{F},S}[n,k]$ be a Reed–Solomon code over a field $\mathbb{F}$ of size $q \geq n$ with $S = (\alpha_1, \alpha_2, \ldots, \alpha_n)$. There is a deterministic algorithm running in time polynomial in q that on input $y \in \mathbb{F}_q^n$ outputs a list of all polynomials $p(X) \in \mathbb{F}[X]$ of degree less than k for which $p(\alpha_i) \neq y_i$ for less than $n - \sqrt{(k-1)n}$ positions $i \in \{1, 2, \ldots, n\}$. Further, at most $O(n^2)$ polynomials will be output by the algorithm in the worst-case.*

Alternatively, one can correct a RS code of block length n and rate $R = k/n$ up to $n - \sqrt{(k-1)}$ errors, or equivalently a fraction $1 - \sqrt{R}$ of errors.

The Reed–Solomon decoding algorithm is based on the solution to the following more general polynomial reconstruction problem which seems like a natural algebraic question in itself. (The problem is more general than RS decoding since the α_i's need not be distinct.)

Problem 1 (Polynomial Reconstruction)
Input: *Integers $k, t \leq n$ and n distinct pairs $\{(\alpha_i, y_i)\}_{i=1}^n$ where $\alpha_i, y_i \in \mathbb{F}$.*
Output: *A list of all polynomials $p(X) \in \mathbb{F}[X]$ of degree less than k which satisfy $p(\alpha_i) = y_i$ for at least t values of $i \in [n]$.*

Theorem 2 *The polynomial reconstruction problem can be solved in time polynomial in $n, |\mathbb{F}|$, provided $t > \sqrt{(k-1)n}$.*

The reader is referred to the original papers [5,9], or a recent survey [1], for details on the above algorithm. A quick, high level peek into the main ideas is given below. The first step in the algorithm consists of an interpolation step where a nonzero bivariate polynomial $Q(X,Y)$ is "fit" through the n pairs (α_i, y_i), so that $Q(\alpha_i, y_i) = 0$ for every i. The key is to do this with relatively low degree; in particular one can find such a $Q(X,Y)$ with so-called $(1, k-1)$-weighted degree at most $D \approx \sqrt{2(k-1)n}$. This degree budget on Q implies that for any polynomial $p(X)$ of degree less than k, $Q(X, p(X))$ will have degree at most D. Now whenever $p(\alpha_i) = y_i$, $Q(\alpha_i, p(\alpha)i)) = Q(\alpha_i, y_i) = 0$. Therefore, if a polynomial $p(X)$ satisfies $p(\alpha_i) = y_i$ for at least t values of i, then $Q(X, p(X))$ has at least t roots. On the other hand the polynomial $Q(X, p(X))$ has degree at most D. Therefore, if $t > D$, one must have $Q(X, p(X)) = 0$, or in other words $Y - p(X)$ is a factor of $Q(X,Y)$. The second step of the algorithm factorized the polynomial $Q(X,Y)$, and all polynomials $p(X)$ that must be output will be found as factors $Y - p(X)$ of $Q(X,Y)$.

Note that since $D \approx \sqrt{2(k-1)n}$ this gives an algorithm for polynomial reconstruction provided the agreement parameter t satisfies $t > \sqrt{2(k-1)n}$ [9]. To get an algorithm for $t > \sqrt{(k-1)n}$, and thus decode beyond half the minimum distance $(n-k)/2$ for all parameter choices for k, n, Guruswami and Sudan [5] use the crucial idea of allowing "multiple roots" in the interpolation step. Specifically, the polynomial Q is required to have $r \geq 1$ roots at each pair (α_i, y_i) for some integer multiplicity parameter r (the notion needs to be formalized properly, see [5] for details). This necessitates an increase in the $(1, k-1)$-weighted degree of a factor of about $r/\sqrt{2}$, but the gain is that one gets a factor r more roots for the polynomial $Q(X, p(X))$. These facts together lead to an algorithm that works as long as $t > \sqrt{(k-1)n}$.

There is an additional significant benefit offered by the multiplicity based decoder. The multiplicities of the interpolation points need not all be equal and they can picked in proportion to the reliability of different received symbols. This gives a powerful way to exploit "soft" information in the decoding stage, leading to impressive coding gains in practice. The reader is referred to the paper by Koetter and Vardy [7] for further details on using multiplicities to encode symbol level reliability information from the channel.

Applications

Reed–Solomon codes have been extensively studied and are widely used in practice. The above decoding algorithm corrects more errors beyond the traditional half the distance limit and therefore directly advances the state of the art on this important algorithmic task. The RS list decoding algorithm has also been the backbone for many further developments in algorithmic coding theory. In particular, using this algorithm in concatenation schemes leads to good binary list-decodable codes. A variant of RS codes called folded RS codes have been used to achieve the optimal trade-off between error-correction radius and rate [3] (see the companion encyclopedia entry by Rudra on folded RS codes).

The RS list decoding algorithm has also found many surprising applications beyond coding theory. In particular, it plays a key role in several results in cryptography and complexity theory (such as constructions of randomness extractors and pseudorandom generators, hardness amplification, constructions to hardcore predicates, traitor tracing, reductions connecting worst-case hardness to average-case hardness, etc.); more information can be found, for instance, in [10] or Chap. 12 in [2].

Open Problems

The most natural open question is whether one can improve the algorithm further and correct more than a fraction $1 - \sqrt{R}$ of errors for RS codes of rate R. It is important to note that there is a combinatorial limitation to the number of errors one can list decode from. One can only list decode in polynomial time from a fraction ρ of errors if for every received word $\mathbf{y}$ the number of RS codewords within distance $e = \rho n$ of $\mathbf{y}$ is bounded by a polynomial function of the block length n. The largest ρ for which this holds as a function of the rate R is called the list decoding radius $\rho_{\text{LD}} = \rho_{\text{LD}}(R)$ of RS codes. The RS list decoding algorithm discussed here implies that $\rho_{\text{LD}}(R) \geq 1 - \sqrt{R}$, and it is trivial to see than $\rho_{\text{LD}}(R) \leq 1 - R$. Are there RS codes (perhaps based on specially structured evaluation points) for which $\rho_{\text{LD}}(R) > 1 - \sqrt{R}$? Are there RS codes for which the $1 - \sqrt{R}$ radius (the so-called "Johnson bound") is actually tight for list decoding? For the more general polynomial reconstruction problem the $\sqrt{(k-1)n}$ agreement cannot be improved upon [4], but this is not known for RS list decoding.

Improving the NP-hardness result of [6] to hold for RS codes over polynomial sized fields and for smaller decoding radii remains an important challenge.

Cross References

► Learning Heavy Fourier Coefficients of Boolean Functions
► List Decoding near Capacity: Folded RS Codes
► LP Decoding

Recommended Reading

1. Guruswami, V.: Algorithmic Results in List Decoding. In: Foundations and Trends in Theoretical Computer Science, vol. 2, issue 2, NOW publishers, Hanover (2007)
2. Guruswami, V.: List Decoding of Error-Correcting Codes. Lecture Notes in Computer Science, vol. 3282. Springer, Berlin (2004)
3. Guruswami, V., Rudra, A.: Explicit codes achieving list decoding capacity: Error-correction with optimal redundancy. IEEE Trans. Inform. Theor. **54**(1), 135–150 (2008)
4. Guruswami, V., Rudra, A.: Limits to list decoding Reed–Solomon codes. IEEE Trans. Inf. Theory. **52**(8), 3642–3649 (2006)
5. Guruswami, V., Sudan, M.: Improved decoding of Reed–Solomon and algebraic-geometric codes. IEEE Trans. Inf. Theory. **45**(6), 1757–1767 (1999)
6. Guruswami, V., Vardy A.: Maximum Likelihood Decoding of Reed–Solomon codes is NP-hard. IEEE Trans. Inf. Theory. **51**(7), 2249–2256 (2005)
7. Koetter, R., Vardy, A.: Algebraic soft-decision decoding of Reed–Solomon codes. IEEE Trans. Inf. Theory. **49**(11), 2809–2825 (2003)

8. Peterson, W.W.: Encoding and error-correction procedures for Bose-Chaudhuri codes. IEEE Trans. Inf. Theory. **6**, 459–470 (1960)
9. Sudan, M.: Decoding of Reed–Solomon codes beyond the error-correction bound. J. Complex. **13**(1), 180–193 (1997)
10. Sudan, M.: List decoding: Algorithms and applications. SIGACT News. **31**(1), 16–27 (2000)

Decremental All-Pairs Shortest Paths

2004; Demetrescu, Italiano

CAMIL DEMETRESCU, GIUSEPPE F. ITALIANO
Department of Information and Computer Systems, University of Rome, Rome, Italy

Keywords and Synonyms

Deletions-only dynamic all-pairs shortest paths

Problem Definition

A dynamic graph algorithm maintains a given property $\mathcal{P}$ on a graph subject to dynamic changes, such as edge insertions, edge deletions and edge weight updates. A dynamic graph algorithm should process queries on property $\mathcal{P}$ quickly, and perform update operations faster than recomputing from scratch, as carried out by the fastest static algorithm. An algorithm is *fully dynamic* if it can handle both edge insertions and edge deletions. A *partially dynamic* algorithm can handle either edge insertions or edge deletions, but not both: it is *incremental* if it supports insertions only, and *decremental* if it supports deletions only.

This entry addressed the *decremental* version of the all-pairs shortest paths problem (APSP), which consists of maintaining a directed graph with real-valued edge weights under an intermixed sequence of the following operations:

delete(u, v): delete edge (u, v) from the graph.
distance(x, y): return the distance from vertex x to vertex y.
path(x, y): report a shortest path from vertex x to vertex y, if any.

A natural variant of this problem supports a generalized delete operation that removes a vertex and all edges incident to it. The algorithms addressed in this entry can deal with this generalized operation within the same bounds.

History of the Problem

A simple-minded solution to this problem would be to rebuild shortest paths from scratch after each deletion using the best static APSP algorithm so that distance and path queries can be reported in optimal time. The fastest known static APSP algorithm for arbitrary real weights has a running time of $O(mn + n^2 \log\log n)$, where m is the number of edges and n is the number of vertices in the graph [13]. This is $\Omega(n^3)$ in the worst case. Fredman [6] and later Takaoka [19] showed how to break this cubic barrier: the best asymptotic bound is by Takaoka, who showed how to solve APSP in $O(n^3 \sqrt{\log\log n / \log n})$ time.

Another simple-minded solution would be to answer queries by running a point-to-point shortest paths computation, without the need to update shortest paths at each deletion. This can be done with Dijkstra's algorithm [3] in $O(m + n \log n)$ time using the Fibonacci heaps of Fredman and Tarjan [5]. With this approach, queries are answered in $O(m + n \log n)$ worst-case time and updates require optimal time.

The dynamic maintenance of shortest paths has a long history, and the first papers date back to 1967 [11,12,17]. In 1985 Even and Gazit [4] presented algorithms for maintaining shortest paths on directed graphs with arbitrary real weights. The worst-case bounds of their algorithm for edge deletions were comparable to recomputing APSP from scratch. Also Ramalingam and Reps [15,16] and Frigioni et al. [7,8] considered dynamic shortest path algorithms with real weights, but in a different model. Namely, the running time of their algorithm is analyzed in terms of the output change rather than the input size (*output bounded complexity*). Again, in the worst case the running times of output-bounded dynamic algorithms are comparable to recomputing APSP from scratch.

The first decremental algorithm that was provably faster than recomputing from scratch was devised by King for the special case of graphs with integer edge weights less than C: her algorithm can update shortest paths in a graph subject to a sequence of $\Omega(n^2)$ deletions in $O(C \cdot n^2)$ amortized time per deletion [9]. Later, Demetrescu and Italiano showed how to deal with graphs with real non-negative edge weights in $O(n^2 \log n)$ amortized time per deletion [2] in a sequence of $\Omega(m/n)$ operations. Both algorithms work in the more general case where edges are not deleted from the graph, but their weight is increased at each update. Moreover, since they update shortest paths explicitly after each deletion, queries are answered in optimal time at any time during a sequence of operations.

Key Results

The decremental APSP algorithm by Demetrescu and Italiano hinges upon the notion of locally shortest paths [2].

Definition 1 A path is *locally shortest* in a graph if all of its proper subpaths are shortest paths.

Notice that by the optimal-substructure property, a shortest path is locally shortest. The main idea of the algorithm is to keep information about locally shortest paths in a graph subject to edge deletions. The following theorem derived from [2] bounds the number of changes in the set of locally shortest paths due to an edge deletion:

Theorem 1 *If shortest paths are unique in the graph, then the number of paths that start or stop being shortest at each deletion is $O(n^2)$ amortized over $\Omega(m/n)$ update operations.*

The result of Theorem 1 is purely combinatorial and assumes that shortest paths are unique in the graph. The latter can be easily achieved using any consistent tie-breaking strategy (see, e. g., [2]). It is possible to design a deletions-only algorithm that pays only $O(\log n)$ time per change in the set of locally shortest paths, using a simple modification of Dijkstra's algorithm [3]. Since by Theorem 1 the amortized number of changes is bounded by $O(n^2)$, this yields the following result:

Theorem 2 *Consider a graph with n vertices and an initial number of m edges subject to a sequence of $\Omega(m/n)$ edge deletions. If shortest paths are unique and edge weights are non-negative, it is possible to support each* `delete` *operation in $O(n^2 \log n)$ amortized time, each* `distance` *query in O(1) worst-case time, and each* `path` *query in $O(\ell)$ worst-case time, where ℓ is the number of vertices in the reported shortest path. The space used is O(mn).*

Applications

Application scenarios of dynamic shortest paths include network optimization [1], document formatting [10], routing in communication systems, robotics, incremental compilation, traffic information systems [18], and dataflow analysis. A comprehensive review of real-world applications of dynamic shortest path problems appears in [14].

URL to Code

An efficient C language implementation of the decremental algorithm addressed in Section "Key Results" is available at the URL: http://www.dis.uniroma1.it/~demetres/experim/dsp.

Cross References

▶ All Pairs Shortest Paths in Sparse Graphs
▶ All Pairs Shortest Paths via Matrix Multiplication
▶ Fully Dynamic All Pairs Shortest Paths

Recommended Reading

1. Ahuja, R., Magnanti, T., Orlin, J.: Network Flows: Theory, Algorithms and Applications. Prentice Hall, Englewood Cliffs, NJ (1993)
2. Demetrescu, C., Italiano, G.: A new approach to dynamic all pairs shortest paths. J. Assoc. Comp. Mach. **51**, 968–992 (2004)
3. Dijkstra, E.: A note on two problems in connexion with graphs. Numerische Mathematik **1**, 269–271 (1959)
4. Even, S., Gazit, H.: Updating distances in dynamic graphs. Meth. Op. Res. **49**, 371–387 (1985)
5. Fredman, M., Tarjan, R.: Fibonacci heaps and their use in improved network optimization algorithms. J. ACM **34**, 596–615 (1987)
6. Fredman, M.L.: New bounds on the complexity of the shortest path problems. SIAM J. Comp. **5**(1), 87–89 (1976)
7. Frigioni, D., Marchetti-Spaccamela, A., Nanni, U.: Semi-dynamic algorithms for maintaining single source shortest paths trees. Algorithmica **22**, 250–274 (1998)
8. Frigioni, D., Marchetti-Spaccamela, A., Nanni, U.: Fully dynamic algorithms for maintaining shortest paths trees. J. Algorithm **34**, 351–381 (2000)
9. King, V.: Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs. In: Proc. 40th IEEE Symposium on Foundations of Computer Science (FOCS'99), pp. 81–99. IEEE Computer Society, New York, USA (1999)
10. Knuth, D., Plass, M.: Breaking paragraphs into lines. Software-Practice Exp. **11**, 1119–1184 (1981)
11. Loubal, P.: A network evaluation procedure. Highway Res. Rec. **205**, 96–109 (1967)
12. Murchland, J.: The effect of increasing or decreasing the length of a single arc on all shortest distances in a graph, tech. rep., LBS-TNT-26, London Business School. Transport Network Theory Unit, London, UK (1967)
13. Pettie, S.: A new approach to all-pairs shortest paths on real-weighted graphs. Theor. Comp. Sci. **312**, 47–74 (2003) special issue of selected papers from ICALP (2002)
14. Ramalingam, G.: Bounded incremental computation. Lect. Note Comp. Sci. 1089 (1996)
15. Ramalingam, G., Reps, T.: An incremental algorithm for a generalization of the shortest path problem. J. Algorithm **21**, 267–305 (1996)
16. Ramalingam, G., Reps, T.: On the computational complexity of dynamic graph problems. Theor. Comp. Sci. **158**, 233–277 (1996)
17. Rodionov, V.: The parametric problem of shortest distances. USSR Comp. Math. Math. Phys. **8**, 336–343 (1968)
18. Schulz, F., Wagner, D., Weihe, K.: Dijkstra's algorithm on-line: an empirical case study from public railroad transport. In: Proc. 3rd Workshop on Algorithm Engineering (WAE'99), pp. 110–123. Notes in Computer Science 1668. London, UK (1999)
19. Takaoka, T.: A new upper bound on the complexity of the all pairs shortest path problem. Inf. Proc. Lett. **43**, 195–199 (1992)

Degree-Bounded Planar Spanner with Low Weight

2005; Song, Li, Wang

WEN-ZHAN SONG[1], XIANG-YANG LI[2] ,
WEIZHAO WANG[3]
[1] School of Engineering and Computer Science, Washington State University, Vancouver, WA, USA
[2] Department of Computer Science, Illinois Institute of Technology, Chicago, IL, USA
[3] Google Inc, Irvine, CA, USA

Keywords and Synonyms

Unified energy-efficient unicast and broadcast topology control

Problem Definition

An important requirement of wireless ad hoc networks is that they should be self-organizing, and transmission ranges and data paths may need to be dynamically restructured with changing topology. Energy conservation and network performance are probably the most critical issues in wireless *ad hoc* networks, because wireless devices are usually powered by batteries only and have limited computing capability and memory. Hence, in such a dynamic and resource-limited environment, each wireless node needs to locally select communication neighbors and adjust its transmission power accordingly, such that all nodes together self-form a topology that is energy efficient for both unicast and broadcast communications.

To support energy-efficient unicast, the topology is preferred to have the following features in the literature:

1. POWER SPANNER: [1,9,13,16,17] Formally speaking, a subgraph H is called a *power spanner* of a graph G if there is a positive real constant ρ such that for any two nodes, the power consumption of the shortest path in H is at most ρ times of the power consumption of the shortest path in G. Here ρ is called the *power stretch factor* or *spanning ratio*.
2. DEGREE BOUNDED: [1,9,11,13,16,17] It is also desirable that the logical node degree in the constructed topology is bounded from above by a small constant. Bounded logical degree structures find applications in Bluetooth wireless networks since a *master* node can have only seven active slaves simultaneously. A structure with small logical node degree will save the cost of updating the routing table when nodes are mobile. A structure with a small degree and using shorter links could improve the overall network throughout [6].
3. PLANAR:[1,4,13,14,16] A network topology is also preferred to be planar (no two edges crossing each other in the graph) to enable some localized routing algorithms to work correctly and efficiently, such as *Greedy Face Routing* (GFG) [2], *Greedy Perimeter Stateless Routing* (GPSR) [5], *Adaptive Face Routing* (AFR) [7], and *Greedy Other Adaptive Face Routing* (GOAFR) [8]. Notice that with planar network topology as the underlying routing structure, these localized routing protocols guarantee the message delivery without using a routing table: each intermediate node can decide which logical neighboring node to forward the packet to using only local information and the position of the source and the destination.

To support energy-efficient broadcast [15], the locally constructed topology is preferred to be *low-weighted* [10,12]: the total link length of the final topology is within a constant factor of that of *EMST*. Recently, several localized algorithms [10,12] have been proposed to construct low-weighted structures, which indeed approximate the energy efficiency of EMST as the network density increases. However, none of them is power efficient for unicast routing.

Before this work, all known topology control algorithms could not support power efficient unicast and broadcast in the same structure. It is indeed challenging to design a unified topology, especially due to the trade off between spanner and low weight property. The main contribution of this algorithm is to address this issue.

Key Results

This algorithm is the *first* localized topology control algorithm for all nodes to maintain a *unified* energy-efficient topology for unicast and broadcast in wireless ad hoc/sensor networks. In one single structure, the following network properties are guaranteed:

1. **Power efficient unicast**: given any two nodes, there is a path connecting them in the structure with total power cost no more than $2\rho + 1$ times the power cost of any path connecting them in the original network. Here $\rho > 1$ is some constant that will be specified later in this algorithm. It assumes that each node u can adjust its power sufficiently to cover its next-hop v on any selected path for unicast.
2. **Power efficient broadcast**: the power consumption for broadcast is within a constant factor of the optimum among all *locally* constructed structures. As proved in [10], to prove this, it equals to prove that the structure is *low-weighted*. Here we called a structure low-weigthed, if its total edge length is within a constant factor of the total length of the Euclidean Minimum Spanning

```
1: First, each node self-constructs the Gabriel graph $GG$ locally. The algorithm to construct $GG$ locally is well-known,
   and a possible implementation may refer to [13]. Initially, all nodes mark themselves WHITE, i. e., unprocessed.
2: Once a WHITE node $u$ has the smallest ID among all its WHITE neighbors in $N(u)$, it uses the following strategy
   to select neighbors:
   1. Node $u$ first sorts all its BLACK neighbors (if available) in $N(u)$ in the distance-increasing order, then sorts
      all its WHITE neighbors (if available) in $N(u)$ similarly. The sorted results are then restored to $N(u)$, by first
      writing the sorted list of BLACK neighbors then appending the sorted list of WHITE neighbors.
   2. Node $u$ scans the sorted list $N(u)$ from left to right. In each step, it keeps the current pointed neighbor $w$ in the
      list, while deletes every conflicted node $v$ in the remainder of the list. Here a node $v$ is conflicted with $w$ means
      that node $v$ is in the $\theta$-dominating region of node $w$. Here $\theta = 2\pi/k$ ($k \geq 9$) is an adjustable parameter.
   Node $u$ then marks itself BLACK, i. e. processed, and notifies each deleted neighboring node $v$ in $N(u)$ by a broad-
   casting message UPDATEN.
3: Once a node $v$ receives the message UPDATEN from a neighbor $u$ in $N(v)$, it checks whether itself is in the nodes
   set for deleting: if so, it deletes the sending node $u$ from list $N(v)$, otherwise, marks $u$ as BLACK in $N(v)$.
4: When all nodes are processed, all selected links $\{uv | v \in N(u), \forall v \in GG\}$ form the final network topology,
   denoted by $S\Theta GG$. Each node can shrink its transmission range as long as it sufficiently reaches its farthest
   neighbor in the final topology.
```

Degree-Bounded Planar Spanner with Low Weight, Algorithm 1
***SΘ GG*: Power-Efficient Unicast Topology**

Tree (EMST). For broadcast or generally multicast, it assumes that each node u can adjust its power sufficiently to cover its farthest down-stream node on any selected structure (typically a tree) for multicast.

3. **Bounded logical node degree**: each node has to communicate with at most $k-1$ logical neighbors, where $k \geq 9$ is an adjustable parameter.
4. **Bounded average physical node degree**: the expected average physical node degree is at most a small constant. Here the physical degree of a node u in a structure H is defined as the number of nodes inside the disk centered at u with radius $\max_{uv \in H} \|uv\|$.
5. **Planar**: there are no edges crossing each other. This enables several localized routing algorithms, such as [2,5,7,8], to be performed on top of this structure and guarantee the packet delivery without using the routing table.
6. **Neighbors Θ-separated**: the directions between any two logical neighbors of any node are separated by at least an angle θ, which reduces the communication interferences.

It is the *first* known *localized* topology control strategy for all nodes together to maintain such a *single* structure with these desired properties. Previously, only a centralized algorithm was reported in [1]. The first step is Algorithm 1 that can construct a power-efficient topology for unicast, then it extends to the final algorithm (Algorithm 2) that can support power-efficient broadcast at the same time.

Definition 1 (θ-Dominating Region) For each neighbor node v of a node u, the *θ-dominating region* of v is the 2θ-cone emanated from u, with the edge uv as its axis.

Let $N_{\text{UDG}}(u)$ be the set of neighbors of node u in UDG, and let $N(u)$ be the set of neighbors of node u in the final topology, which is initialized as the set of neighbor nodes in GG.

Algorithm 1 constructs a degree-$(k-1)$ planar power spanner.

Lemma 1 *Graph $S\Theta GG$ is connected if the underlying graph GG is connected. Furthermore, given any two nodes u and v, there exists a path $\{u, t_1, \ldots, t_r, v\}$ connecting them such that all edges have length less than $\sqrt{2}\|uv\|$.*

Theorem 2 *The structure $S\Theta GG$ has node degree at most $k-1$ and is planar power spanner with neighbors Θ-separated. Its power stretch factor is at most $\rho = \sqrt{2}^{\beta} / (1 - (2\sqrt{2} \sin \frac{\pi}{k})^{\beta})$, where $k \geq 9$ is an adjustable parameter.*

Obviously, the construction is consistent for two endpoints of each edge: if an edge uv is kept by node u, then it is also kept by node v. It is worth mentioning that, the number 3 in criterion $\|xy\| > \max(\|uv\|, 3\|ux\|, 3\|vy\|)$ is carefully selected.

Theorem 3 *The structure $LS\Theta GG$ is a degree-bounded planar spanner. It has a constant power spanning ratio*

```
1: All nodes together construct the graph $S\Theta GG$ in a localized manner, as described in Algorithm 1. Then, each
   node marks its incident edges in $S\Theta GG$ *unprocessed*.
2: Each node $u$ locally broadcasts its incident edges in $S\Theta GG$ to its one-hop neighbors and listens to its neighbors.
   Then, each node $x$ can learn the existence of the set of 2-hop links $E_2(x)$, which is defined as follows: $E_2(x) = \{uv \in S\Theta GG \mid u \text{ or } v \in N_{UDG}(x)\}$.
   In other words, $E_2(x)$ represents the set of edges in $S\Theta GG$ with at least one endpoint in the transmission range of node $x$.
3: Once a node $x$ learns that its *unprocessed* incident edge $xy$ has the smallest ID among all *unprocessed* links in
   $E_2(x)$, it will delete edge $xy$ if there exists an edge $uv \in E_2(x)$ (here both $u$ and $v$ are different from $x$ and $y$),
   such that $\|xy\| > \max(\|uv\|, 3\|ux\|, 3\|vy\|)$; otherwise it simply marks edge $xy$ *processed*. Here assume that
   $uvyx$ is the convex hull of $u$, $v$, $x$ and $y$. Then the link status is broadcasted to all neighbors through a message
   UPDATESTATUS(XY).
4: Once a node $u$ receives a message UPDATESTATUS(XY), it records the status of link $xy$ at $E_2(u)$.
5: Each node repeats the above two steps until all edges have been *processed*. Let $LS\Theta GG$ be the final structure
   formed by all remaining edges in $S\Theta GG$.
```

Degree-Bounded Planar Spanner with Low Weight, Algorithm 2
Construct *LSΘ GG*: Planar Spanner with Bounded Degree and Low Weight

$2\rho + 1$, *where ρ is the power spanning ratio of $S\Theta GG$. The node degree is bounded by $k - 1$ where $k \geq 9$ is a customizable parameter in $S\Theta GG$.*

Theorem 4 *The structure $LS\Theta GG$ is low-weighted.*

Theorem 5 *Assuming that both the ID and the geometry position can be represented by $\log n$ bits each, the total number of messages during constructing the structure $LS\Theta GG$ is in the range of $[5n, 13n]$, where each message has at most $O(\log n)$ bits.*

Compared with previous known low-weighted structures [10,12], $LS\Theta GG$ not only achieves more desirable properties, but also costs much less messages during construction. To construct $LS\Theta GG$, each node only needs to collect the information $E_2(x)$ which costs at most $6n$ messages for n nodes. The Algorithm 2 can be generally applied to any known degree-bounded planar spanner to make it low-weighted while keeping all its previous properties, except increasing the spanning ratio from ρ to $2\rho + 1$ theoretically.

In addition, the expected average node interference in the structure is bounded by a small constant. This is significant on its own due to the following reasons: it has been taken for granted that "*a network topology with small logical node degree will guarantee a small interference*" and recently Burkhart et al. [3] showed that this is not true generally. This work also shows that, although generally a small logical node degree cannot guarantee a small interference, the expected average interference is indeed small if the logical communication neighbors are chosen carefully.

Theorem 6 *For a set of nodes produced by a Poisson point process with density n, the expected maximum node interferences of EMST, GG, RNG, and Yao are at least $\Theta(\log n)$.*

Theorem 7 *For a set of nodes produced by a Poisson point process with density n, the expected average node interferences of EMST are bounded from above by a constant.*

This result also holds for nodes deployed with uniform random distribution.

Applications

Localized topology control in wireless ad hoc networks are critical mechanisms to maintain network connectivity and provide feedback to communication protocols. The major traffic in networks are unicast communications. There is a compelling need to conserve energy and improve network performance by maintaining an energy-efficient topology in localized ways. This algorithm achieves this by choosing relatively smaller power levels and size of communication neighbors for each node (e. g., reducing interference). Also, broadcasting is often necessary in MANET routing protocols. For example, many unicast routing protocols such as Dynamic Source Routing (DSR), Ad Hoc On Demand Distance Vector (AODV), Zone Routing Protocol (ZRP), and Location Aided Routing (LAR) use broadcasting or a derivation of it to establish routes. It is highly important to use power-efficient broadcast algorithms for such networks since wireless devices are often powered by batteries only.

Cross References

► Applications of Geometric Spanner Networks
► Geometric Spanners
► Planar Geometric Spanners
► Sparse Graph Spanners

Recommended Reading

1. Bose, P., Gudmundsson, J., Smid, M.: Constructing plane spanners of bounded degree and low weight. In: Proceedings of European Symposium of Algorithms, University of Rome, 17–21 September 2002
2. Bose, P., Morin, P., Stojmenovic, I., Urrutia, J.: Routing with guaranteed delivery in ad hoc wireless networks. ACM/Kluwer Wireless Networks **7**(6), 609–616 (2001). 3rd int. Workshop on Discrete Algorithms and methods for mobile computing and communications, 48–55 (1999)
3. Burkhart, M., von Rickenbach, P., Wattenhofer, R., Zollinger, A.: Does topology control reduce interference. In: ACM Int. Symposium on Mobile Ad-Hoc Networking and Computing (MobiHoc), Tokyo, 24–26 May 2004
4. Gabriel, K.R., Sokal, R.R.: A new statistical approach to geographic variation analysis. Syst. Zool. **18**, 259–278 (1969)
5. Karp, B., Kung, H.T.: Gpsr: Greedy perimeter stateless routing for wireless networks. In: Proc. of the ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom), Boston, 6–11 August 2000
6. Kleinrock, L., Silvester, J.: Optimum transmission radii for packet radio networks or why six is a magic number. In: Proceedings of the IEEE National Telecommunications Conference, pp. 431–435, Birmingham, 4–6 December 1978
7. Kuhn, F., Wattenhofer, R., Zollinger, A.: Asymptotically optimal geometric mobile ad-hoc routing. In: International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DIALM), Atlanta, 28 September 2002
8. Kuhn, F., Wattenhofer, R., Zollinger, A.: Worst-case optimal and average-case efficient geometric ad-hoc routing. In: ACM Int. Symposium on Mobile Ad-Hoc Networking and Computing (MobiHoc) Anapolis, 1–3 June 2003
9. Li, L., Halpern, J.Y., Bahl, P., Wang, Y.-M., Wattenhofer, R.: Analysis of a cone-based distributed topology control algorithms for wireless multi-hop networks. In: PODC: ACM Symposium on Principle of Distributed Computing, Newport, 26–29 August 2001
10. Li, X.-Y.: Approximate MST for UDG locally. In: COCOON, Big Sky, 25–28 July 2003
11. Li, X.-Y., Wan, P.-J., Wang, Y., Frieder, O.: Sparse power efficient topology for wireless networks. In: IEEE Hawaii Int. Conf. on System Sciences (HICSS), Big Island, 7–10 January 2002
12. Li, X.-Y., Wang, Y., Song, W.-Z., Wan, P.-J., Frieder, O.: Localized minimum spanning tree and its applications in wireless ad hoc networks. In: IEEE INFOCOM, Hong Kong, 7–11 March 2004
13. Song, W.-Z., Wang, Y., Li, X.-Y. Frieder, O.: Localized algorithms for energy efficient topology in wireless ad hoc networks. In: ACM Int. Symposium on Mobile Ad-Hoc Networking and Computing (MobiHoc), Tokyo, 24–26 May 2004
14. Toussaint, G.T.: The relative neighborhood graph of a finite planar set. Pattern Recognit. **12**(4), 261–268 (1980)
15. Wan, P.-J., Calinescu, G., Li, X.-Y., Frieder, O.: Minimum-energy broadcast routing in static ad hoc wireless networks. ACM Wireless Networks (2002), To appear, Preliminary version appeared in IEEE INFOCOM, Anchorage, 22–26 April 2001
16. Wang, Y., Li, X.-Y.: Efficient construction of bounded degree and planar spanner for wireless networks. In: ACM DIALM-POMC Joint Workshop on Foundations of Mobile Computing, San Diego, 19 September 2003
17. Yao, A.C.-C.: On constructing minimum spanning trees in k-dimensional spaces and related problems. SIAM J. Comput. **11**, 721–736 (1982)

Degree-Bounded Trees

1994; Fürer, Raghavachari

MARTIN FÜRER
Department of Computer Science and Engineering, The Pennsylvania State University, University Park, PA, USA

Keywords and Synonyms

Bounded degree spanning trees; Bounded degree Steiner trees

Problem Definition

The problem is to construct a spanning tree of small degree for a connected undirected graph $G = (V, E)$. In the Steiner version of the problem, a set of distinguished vertices $D \subseteq V$ is given along with the input graph G. A Steiner tree is a tree in G which spans at least the set D.

As finding a spanning or Steiner tree of the smallest possible degree Δ^* is NP-hard, one is interested in approximating this minimization problem. For many such combinatorial optimization problems, the goal is to find an approximation in polynomial time (a constant or larger factor). For the spanning and Steiner tree problems, the iterative polynomial time approximation algorithms of Fürer and Raghavachari [8] (see also [14]) find much better solutions. The degree Δ of the solution tree is at most $\Delta^* + 1$.

There are very few natural NP-hard optimization problems for which the optimum can be achieved up to an additive term of 1. One such problem is coloring a planar graph, where coloring with four colors can be done in polynomial time. On the other hand, 3-coloring is NP-complete even for planar graphs. An other such problem is edge coloring a graph of degree Δ. While coloring with $\Delta + 1$ colors is always possible in polynomial time, Δ edge coloring is NP-complete.

Chvátal [3] has defined the *toughness* $\tau(G)$ of a graph as the minimum ratio $|X|/c(X)$ such that the subgraph of G induced by $V \backslash X$ has $c(X) \geq 2$ connected components. The inequality $1/\tau(G) \leq \Delta^*$ immediately follows. Win [17] has shown that $\Delta^* < \frac{1}{\tau(G)} + 3$; i. e., the inverse of the toughness is actually a good approximation of Δ^*.

A set X, such that the ratio $|X|/c(X)$ is the toughness $\tau(G)$, can be viewed as witnessing the upper bound $|X|/c(X)$ on $\tau(G)$ and therefore the lower bound $c(X)/|X|$ on Δ^*. Strengthening this notion, Fürer and Raghavachari [8] define X to be a witness set for $\Delta^* \geq d$ if d is the smallest integer greater or equal to $(|X| + c(X) - 1)/|X|$. Their algorithm not only outputs a spanning tree, but also a witness set X, proving that its degree is at most $\Delta^* + 1$.

Key Results

The minimum degree spanning tree and Steiner tree problems are easily seen to be NP-hard, as they contain the Hamiltonian path problem. Hence, we cannot expect a polynomial time algorithm to find a solution of minimal possible degree Δ^*. The same argument also shows that an approximation by a factor less than 3/2 is impossible in polynomial time unless $P = NP$.

Initial approximation algorithms obtained solutions of degree $O(\Delta^* \log n)$ [6], where $n = |V|$ is the number of vertices. The optimal result for the spanning tree case has been obtained by Fürer and Raghavachari [7, 8].

Theorem 1 *Let Δ^* be the degree of an unknown minimum degree spanning tree of an input graph $G = (V, E)$. There is a polynomial time approximation algorithm for the minimum degree spanning tree problem that finds a spanning tree of degree at most $\Delta^* + 1$.*

Later this result has been extended to the Steiner tree case [8].

Theorem 2 *Assume a Steiner tree problem is defined by a graph $G = (V, E)$ and an arbitrary subset D of vertices V. Let Δ^* be the degree of an unknown minimum degree Steiner tree of G spanning at least the set D. There is a polynomial time approximation algorithm for the minimum degree Steiner tree problem that finds a Steiner tree of degree at most $\Delta^* + 1$.*

Both approximation algorithms run in time $O(mn \cdot \log n\, \alpha(m, n))$, where m is the number of edges and α is the inverse Ackermann function.

Applications

Some possible direct applications are in networks for noncritical broadcasting, where it might be desirable to bound the load per node, and in designing power grids, where the cost of splitting increases with the degree. Another major benefit of a small degree network is limiting the effect of node failure.

Furthermore, the main results on approximating the minimum degree spanning and Steiner tree problems have been the basis for approximating various network design problems, sometimes involving additional parameters.

Klein, Krishnan, Raghavachari and Ravi [11] find 2-connected subgraphs of approximately minimal degree in 2-connected graphs, as well as approximately minimal degree spanning trees (branchings) in directed graphs. Their algorithms run in quasi-polynomial time, and approximate the degree Δ^* by $(1 + \epsilon)\Delta^* + O(\log_{1+\epsilon} n)$.

Often the goal is to find a spanning tree that simultaneously has a small degree and a small weight. For a graph having an minimum weight spanning tree (MST) of degree Δ^* and weight w, Fischer [5] finds a spanning tree with degree $O(\Delta^* + \log n)$ and weight w, (i. e., an MST of small weight) in polynomial time.

Könemann and Ravi [12,13] provide a bi-criteria approximation. For a given $B^* \geq \Delta^*$, let w be the minimum weight of any spanning tree of degree at most B^*. The polynomial time algorithm finds a spanning tree of degree $O(B^* + \log n)$ and weight $O(w)$. In the second paper, the algorithm adapts to the case of a different degree bound on each vertex. Chaudhuri et al. [2] further improved this result to approximate both the degree B^* and the weight w by a constant factor.

In another extension of the minimum degree spanning tree problem, Ravi and Singh [15] have obtained a strict generalization of the $\Delta^* + 1$ spanning tree approximation [8]. Their polynomial time algorithm finds an MST of degree $\Delta^* + k$ for the case of a graph with k distinct weights on the edges.

Recently, there have been some drastic improvements. Again, let w be the minimum cost of a spanning tree of given degree B^*. Goemans [9] obtains a spanning tree of cost w and degree $B^* + 2$. Finally, Singh and Lau [16] decrease the degree to $B^* + 1$ and also handle individual degree bounds Δ_v^* for each vertex v in the same way.

Interesting approximation algorithms are also known for the 2-dimensional Euclidian minimum weight bounded degree spanning tree problem, where the vertices are points in the plane and edge weights are the Euclidian distances. Khuller, Raghavachari, and Young [10] show factor 1.5 and 1.25 approximations for degree bounds 3 and 4 respectively. These bounds have later been improved slightly by Chan [1]. Slightly weaker results are obtained by Fekete et al. [4], using flow-based methods, for the more general case where the weight function just satisfies the triangle inequality.

Open Problems

The time complexity of the minimum degree spanning and Steiner tree algorithms [8] is $O(mn\,\alpha(m,n)\log n)$. Can it be improved to $O(mn)$? In particular, what can be gained by initially selecting a reasonable Steiner tree with some greedy technique instead of starting the iteration with an arbitrary Steiner tree?

Is there an efficient parallel algorithm that can obtain a $\Delta^* + 1$ approximation in poly-logarithmic time? Fürer and Raghavachari [6] have obtained such an NC-algorithm, but only with a factor $O(\log n)$ approximation of the degree.

Cross References

► Fully Dynamic Connectivity
► Graph Connectivity
► Minimum Energy Cost Broadcasting in Wireless Networks
► Minimum Spanning Trees
► Steiner Forest
► Steiner Trees

Recommended Reading

1. Chan, T.M.: Euclidean bounded-degree spanning tree ratios. Discret. Comput. Geom. **32**(2), 177–194 (2004)
2. Chaudhuri, K., Rao, S., Riesenfeld, S., Talwar, K.: A push-relabel algorithm for approximating degree bounded MSTs. In: Proceedings of the 33rd International Colloquium on Automata, Languages and Programming (ICALP 2006), Part I. LNCS, vol. 4051, pp. 191–201. Springer, Berlin (2006)
3. Chvátal, V.: Tough graphs and Hamiltonian circuits. Discret. Math. **5**, 215–228 (1973)
4. Fekete, S.P., Khuller, S., Klemmstein, M., Raghavachari, B., Young, N.: A network-flow technique for finding low-weight-bounded-degree spanning trees. In: Proceedings of the 5th Integer Programming and Combinatorial Optimization Conference (IPCO 1996) and J. Algorithms **24**(2), 310–324 (1997)
5. Fischer, T.: Optimizing the degree of minimum weight spanning trees, Technical Report TR93–1338. Cornell University, Computer Science Department (1993)
6. Fürer, M., Raghavachari, B.: An NC approximation algorithm for the minimum-degree spanning tree problem. In: Proceedings of the 28th Annual Allerton Conference on Communication, Control and Computing, 1990, pp. 174–281
7. Fürer, M., Raghavachari, B.: Approximating the minimum degree spanning tree to within one from the optimal degree. In: Proceedings of the Third Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 1992), 1992, pp. 317–324
8. Fürer, M., Raghavachari, B.: Approximating the minimum-degree Steiner tree to within one of optimal. J. Algorithms **17**(3), 409–423 (1994)
9. Goemans, M.X.: Minimum bounded degree spanning trees. In: Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006), 2006, pp. 273–282
10. Khuller, S., Raghavachari, B., Young, N.: Low-degree spanning trees of small weight. SIAM J. Comput. **25**(2), 355–368 (1996)
11. Klein, P.N., Krishnan, R., Raghavachari, B., Ravi, R.: Approximation algorithms for finding low-degree subgraphs. Networks **44**(3), 203–215 (2004)
12. Könemann, J., Ravi, R.: A matter of degree: Improved approximation algorithms for degree-bounded minimum spanning trees. SIAM J. Comput. **31**(6), 1783–1793 (2002)
13. Könemann, J., Ravi, R.: Primal-dual meets local search: Approximating MSTs with nonuniform degree bounds. SIAM J. Comput. **34**(3), 763–773 (2005)
14. Raghavachari, B.: Algorithms for finding low degree structures. In: Hochbaum, D.S. (ed.) Approximation Algorithms for NP-Hard Problems. pp. 266–295. PWS Publishing Company, Boston (1995)
15. Ravi, R., Singh, M.: Delegate and conquer: An LP-based approximation algorithm for minimum degree MSTs. In: Proceedings of the 33rd International Colloquium on Automata, Languages and Programming (ICALP 2006) Part I. LNCS, vol. 4051, pp. 169–180. Springer, Berlin (2006)
16. Singh, M., Lau, L.C.: Approximating minimum bounded degree spanning trees to within one of optimal. In: Proceedings of the thirty-ninth Annual ACM Symposium on Theory of Computing (STOC 2007), New York, NY, 2007, pp. 661–670
17. Win, S.: On a connection between the existence of *k*-trees and the toughness of a graph. Graphs Comb. **5**(1), 201–205 (1989)

Deterministic Broadcasting in Radio Networks

2000; Chrobak, Gąsieniec, Rytter

LESZEK GĄSIENIEC
Department of Computer Science,
University of Liverpool, Liverpool, UK

Keywords and Synonyms

Wireless networks; Dissemination of information; One-to-all communication

Problem Definition

One of the most fundamental communication problems in wired as well as wireless networks is **broadcasting**, where one distinguished source node has a message that needs to be sent to all other nodes in the network.

The radio network abstraction captures the features of distributed communication networks with multi-access channels, with minimal assumptions on the channel model and processors' knowledge. Directed edges model uni-directional links, including situations in which one of two adjacent transmitters is more powerful than the other. In particular, there is no feedback mechanism (see, for example, [13]). In some applications, collisions may be difficult to distinguish from the noise that is normally present on the channel, justifying the need for protocols that do not depend on the reliability of the collision detection mechanism (see [9,10]). Some network configurations are subject to frequent changes. In other networks, topologies could be unstable or dynamic; for example, when mobile users are present. In such situations, algorithms that do not assume any specific topology are more desirable.

More formally a radio network is a directed graph where by n we denote the number of nodes in this graph. If there is an edge from u to v, then we say that v is an *out-neighbor* of u and u is an *in-neighbor* of v. Each node is assigned a unique identifier from the set $\{1, 2, \dots, n\}$. In the broadcast problem, one node, for example node 1, is distinguished as the *source node*. Initially, the nodes do not possess any other information. In particular, they do not know the network topology.

The time is divided into discrete time steps. All nodes start simultaneously, have access to a common clock, and work synchronously. A broadcasting algorithm is a protocol that for each identifier *id*, given all past messages received by *id*, specifies, for each time step t, whether *id* will transmit a message at time t, and if so, it also specifies the message. A message M transmitted at time t from a node u is sent instantly to all its out-neighbors. An out-neighbor v of u receives M at time step t only if no collision occurred, that is, if the other in-neighbors of v do not transmit at time t at all. Further, collisions cannot be distinguished from background noise. If v does not receive any message at time t, it knows that either none of its in-neighbors transmitted at time t, or that at least two did, but it does not know which of these two events occurred. The *running time* of a broadcasting algorithm is the smallest t such that for any network topology, and any assignment of identifiers to the nodes, all nodes receive the source message no later than at step t.

All efficient radio broadcasting algorithms are based on the following purely combinatorial concept of selectors.

Selectors Consider subsets of $\{1, \dots, n\}$. We say that a set S *hits* a set X iff $|S \cap X| = 1$, and that S *avoids* Y iff $S \cap Y = \emptyset$. A family $\mathcal{S}$ of sets is a *w-selector* if it satisfies the following property:

(∗) For any two disjoint sets X, Y with $w/2 \le |X| \le w$, $|Y| \le w$, there is a set in $\mathcal{S}$ which hits X and avoids Y.

A complete layered network is a graph consisting of layers $L_0, \dots, L_{m-1}$, in which each node in layer L_i is directly connected to every node in layer L_{i+1}, for all $i = 0, \dots, m-1$. The layer L_0 contains only the source node s.

Key Results

Theorem 1 ([5]) *For all positive integers w and n, s.t., $w \le n$ there exists a w-selector $\bar{S}$ with $O(w \log n)$ sets.*

Theorem 2 ([5]) *There exists a deterministic $O(n \log^2 n)$-time algorithm for broadcasting in radio networks with arbitrary topology.*

Theorem 3 ([5]) *There exists a deterministic $O(n \log n)$-time algorithm for broadcasting in complete layered radio networks.*

Applications

Prior to this work, Bruschi and Del Pinto showed in [1] that radio broadcasting requires time $\Omega(n \log D)$ in the worst case. In [2], Chlebus et al. presented a broadcasting algorithm with time complexity $O(n^{11/6})$ – the first subquadratic upper bound. This upper bound was later improved to $O(n^{5/3} \log^3 n)$ by De Marco and Pelc [8], and by Chlebus et al. [3] to $O(n^{3/2})$ by application of finite geometries.

Recently, Kowalski and Pelc in [12] proposed a faster $O(n \log n \log D)$–time radio broadcasting algorithm, where D is the eccentricity of the network. Later, Czumaj and Rytter showed in [6] how to reduce this bound to $O(n \log^2 D)$. The results presented in [5], see Theorems 1, 2, and 3, as well as further improvements in [6,12] are existential (non-constructive). The proofs are based on the probabilistic method. A discussion on efficient explicit construction of selectors was initiated by Indyk in [11], and then continued by Chlebus and Kowalski in [4].

More careful analysis and further discussion on selectors in the context of *combinatorial group testing* can be found in [7], where DeBonis et al. proved that the size of selectors is $\Theta(w \log \frac{n}{w})$.

Open Problems

The exact complexity of radio broadcasting remains an open problem, although the gap between the lower and upper bounds $\Omega(n \log D)$ and $O(n \log^2 D)$ is now only a factor of $\log D$. Another promising direction for further studies is improvement of efficient explicit construction of selectors.

Recommended Reading

1. Bruschi, D., Del Pinto, M.: Lower Bounds for the Broadcast Problem in Mobile Radio Networks. Distrib. Comput. **10**(3), 129–135 (1997)
2. Chlebus, B.S., Gąsieniec, L., Gibbons, A.M., Pelc, A., Rytter, W.: Deterministic broadcasting in unknown radio networks. Distrib. Comput. **15**(1), 27–38 (2002)
3. Chlebus, M., Gąsieniec, L., Östlin, A., Robson, J.M.: Deterministic broadcasting in radio networks. In: Proc. 27th International Colloquium on Automata, Languages and Programming.LNCS, vol. 1853, pp. 717–728, Geneva, Switzerland (2000)
4. Chlebus, B.S., Kowalski, D.R.: Almost Optimal Explicit Selectors. In: Proc. 15th International Symposium on Fundamentals of Computation Theory, pp. 270–280, Lübeck, Germany (2005)
5. Chrobak, M., Gąsieniec, L., Rytter, W.: Fast Broadcasting and Gossiping in Radio Networks,. In: Proc. 41st Annual Symposium on Foundations of Computer Science, pp. 575–581, Redondo Beach, USA (2000) Full version in J. Algorithms **43**(2) 177–189 (2002)
6. Czumaj, A., Rytter, W.: Broadcasting algorithms in radio networks with unknown topology. J. Algorithms **60**(2), 115–143 (2006)
7. De Bonis, A., Gąsieniec, L., Vaccaro, U.: Optimal Two-Stage Algorithms for Group Testing Problems. SIAM J. Comput. **34**(5), 1253–1270 (2005)
8. De Marco, G., Pelc, A.: Faster broadcasting in unknown radio networks. Inf. Process. Lett. **79**(2), 53–56 (2001)
9. Ephremides, A., Hajek, B.: Information theory and communication networks: an unconsummated union. IEEE Trans. Inf. Theor. **44**, 2416–2434 (1998)
10. Gallager, R.: A perspective on multiaccess communications. IEEE Trans. Inf. Theor. **31**, 124–142 (1985)
11. Indyk, P.: Explicit constructions of selectors and related combinatorial structures, with applications. In: Proc. 13th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 697–704, San Francisco, USA (2002)
12. Kowalski, D.R., Pelc, A.: Broadcasting in undirected ad hoc radio networks. Distr. Comput. **18**(1), 43–57 (2005)
13. Massey, J.L., Mathys, P.: The collision channel without feedback. IEEE Trans. Inf. Theor. **31**, 192–204 (1985)

Deterministic Searching on the Line

1988; Baeza-Yates, Culberson, Rawlins

RICARDO BAEZA-YATES
Department of Computer Science,
University of Chile,
Santiago, Chile

Keywords and Synonyms

Searching for a point in a line; Searching in one dimension; Searching for a line (or a plane) of known slope in the plane (or a 3D space)

Problem Definition

The problem is to design a strategy for a searcher (or a number of searchers) located initially at some start point on a line to reach an unknown target point. The target point is detected only when a searcher is located on it. There are several variations depending on the information about the target point, how many parallel searchers are available and how they can communicate, and the type of algorithm. The cost of the search algorithm is defined as the distance traveled until finding the point relative to the distance of the starting point to the target. This entry only covers deterministic algorithms.

Key Results

Consider just one searcher. If one knows the direction to the target, the solution is trivial and the relative cost is 1. If one knows the distance to the target, the solution is also simple. Walk that distance to one side and if the target is not found, go back and travel to the other side until the target is found. In the worst case the cost of this algorithm is 3.

If no information is known about the target, the solution is not trivial. The optimal algorithm follows a linear logarithmic spiral with exponent 2 and has cost 9 plus lower order terms. That is, one takes $1, 2, 4, 8, \ldots, 2^i, \ldots$ steps to each side in an alternating fashion, each time returning to the origin, until the target is found. This result was first discovered by Gal and rediscovered independently by Baeza-Yates et al.

If one has more searchers, say m, the solution is trivial if they have instantaneous communication. Two searchers walk in opposite directions and the rest stay at the origin. The searcher that finds the target communicates this to all the others. Hence, the cost for all searchers is $m + 2$, assuming that all of them must reach the target. If they do not have communication the solution is more complicated and the optimal algorithm is still an open problem.

The searching setting can also be changed, like finding a point in a set of r rays, where the optimal algorithm has cost $1 + 2r^r/(r-1)^{r-1}$, which tends to $1 + 2e \approx 6.44$.

Other variations are possible. For example, if one is interested in the average case one can have a probability distribution for finding the target point, obtaining paradoxical results, as an optimal finite distance algorithm with an infinite number of turning points. On the other hand, in the worst case, if there is a cost d associated with each turn, the optimal distance is $9\ \mathrm{OPT} + 2d$, where OPT is the distance between the origin and the target. This last case has also been solved for r rays.

The same ideas of doubling in each step can be extended to find a target point in an unknown simple polygon or to find a line with known slope in the plane. The same spiral search can also be used to find an arbitrary line in the plane with cost 13.81. The optimality of this result is still an open problem.

Applications

This problem is a basic element for robot navigation in unknown environments. For example, it arises when a robot needs to find where a wall ends, if the robot can only sense the wall but not see it.

Cross References

► Randomized Searching on Rays or the Line

Recommended Reading

1. Alpern, S., Gal, S.: The Theory of Search Games and Rendevouz. Kluwer Academic Publishers, Dordrecht (2003)
2. Baeza-Yates, R., Culberson, J., Rawlins, G.: Searching in the Plane. Inf. Comput. **106**(2), 234–252 (1993) Preliminary version as Searching with uncertainty. In: Karlsson, R., Lingas, A. (eds.) Proceedings SWAT 88, First Scandinavian Workshop on Algorithm Theory. Lecture Notes in Computer Science, vol. 318, pp. 176–189. Halmstad, Sweden (1988)
3. Baeza-Yates, R., Schott, R.: Parallel searching in the plane. Comput. Geom. Theor. Appl. **5**, 143–154 (1995)
4. Blum, A., Raghavan, P., Schieber, B.: Navigating in Unfamiliar Geometric Terrain. In: On Line Algorithms, pp. 151–155, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, Providence RI (1992) Preliminary Version in STOC 1991, pp. 494–504
5. Demaine, E., Fekete, S., Gal, S.: Online searching with turn cost. Theor. Comput. Sci. **361**, 342–355 (2006)
6. Gal, S.: Minimax solutions for linear search problems. SIAM J. Appl. Math. **27**, 17–30 (1974)
7. Gal, S.: Search Games, pp. 109–115, 137–151, 189–195. Academic Press, New York (1980)
8. Hipke, C., Icking, C., Klein, R., Langetepe, E.: How to Find a point on a line within a Fixed distance. Discret. Appl. Math. **93**, 67–73 (1999)
9. Kao, M.-Y., Reif, J.H., Tate, S.R.: Searching in an unknown environment: an optimal randomized algorithm for the cow-path problem. Inf. Comput. **131**(1), 63–79 (1996) Preliminary version in SODA '93, pp. 441–447
10. Lopez-Ortiz, A.: On-Line Target Searching in Bounded and Unbounded Domains: Ph. D. Thesis, Technical Report CS-96-25, Dept. of Computer Sci., Univ. of Waterloo (1996)
11. Lopez-Ortiz, A., Schuierer, S.: The Ultimate Strategy to Search on m Rays? Theor. Comput. Sci. **261**(2), 267–295 (2001)
12. Papadimitriou, C.H., Yannakakis, M.: Shortest Paths without a Map. Theor. Comput. Sci. **84**, 127–150 (1991) Preliminary version in ICALP '89
13. Schuierer, S.: Lower bounds in on-line geometric searching. Comput. Geom. **18**, 37–53 (2001)

Detour

► Dilation of Geometric Networks
► Geometric Dilation of Geometric Networks
► Planar Geometric Spanners

Dictionary-Based Data Compression

1977; Ziv, Lempel

TRAVIS GAGIE, GIOVANNI MANZINI
Department of Computer Science,
University of Eastern Piedmont, Alessandria, Italy

Keywords and Synonyms

LZ compression; Ziv–Lempel compression; Parsing-based compression

Problem Definition

The problem of lossless data compression is the problem of compactly representing data in a format that admits the faithful recovery of the original information. Lossless data compression is achieved by taking advantage of the redundancy which is often present in the data generated by either humans or machines.

Dictionary-based data compression has been "the solution" to the problem of lossless data compression for nearly 15 years. This technique originated in two theoretical papers of Ziv and Lempel [15,16] and gained popularity in the "80s" with the introduction of the Unix tool `compress` (1986) and of the gif image format (1987). Although today there are alternative solutions to the problem of lossless data compression (e. g., Burrows-Wheeler compression and Prediction by Partial Matching), dictionary-based compression is still widely used in everyday applications: consider for example the zip utility and its variants, the modem compression standards V.42bis and V.44, and the transparent compression of pdf documents. The main reason for the success of dictionary-based compression is its unique combination of compression power and compression/decompression speed. The reader should refer to [13] for a review of several dictionary-based compression algorithms and of their main features.

Key Results

Let T be a string drawn from an alphabet Σ. Dictionary-based compression algorithms work by parsing the input into a sequence of substrings (also called words) $T_1, T_2, \ldots, T_d$ and by encoding a compact representation of these substrings. The parsing is usually done incrementally and on-line with the following iterative procedure.

Assume the encoder has already parsed the substrings $T_1, T_2, \ldots, T_{i-1}$. To proceed, the encoder maintains a dictionary of potential candidates for the next word T_i and associates a unique codeword with each of them. Then, it looks at the incoming data, selects one of the candidates, and emits the corresponding codeword. Different algorithms use different strategies for establishing which words are in the dictionary and for choosing the next word T_i. A larger dictionary implies a greater flexibility for the choice of the next word, but also longer codewords. Note that for efficiency reasons the dictionary is usually not built explicitly: the whole process is carried out implicitly using appropriate data structures.

Dictionary-based algorithms are usually classified into two families whose respective ancestors are two parsing strategies, both proposed by Ziv and Lempel and today universally known as LZ78 [16] and LZ77 [15].

The LZ78 Algorithm

Assume the encoder has already parsed the words $T_1, T_2, \ldots, T_{i-1}$, that is, $T = T_1 T_2 \cdots T_{i-1} \hat{T}_i$ for some text suffix $\hat{T}_i$. The LZ78 dictionary is defined as the set of strings obtained by adding a single character to one of the words $T_1, \ldots, T_{i-1}$ or to the empty word. The next word T_i is defined as the longest prefix of $\hat{T}_i$ which is a dictionary word. For example, for $T = aabbaaabaabaabba$ the LZ78 parsing is: a, ab, b, aa, aba, $abaa$, bb, a. It is easy to see that all words in the parsing are distinct, with the possible exception of the last one (in the example the word a). Let T_0 denote the empty word. If $T_i = T_j\alpha$, with $0 \le j < i$ and $\alpha \in \Sigma$, the codeword emitted by LZ78 for T_i will be the pair (j, α). Thus, if LZ78 parses the string T into t words, its output will be bounded by $t \log t + t \log |\Sigma| + \Theta(t)$ bits.

The LZ77 Algorithm

Assume the encoder has already parsed the words $T_1, T_2, \ldots, T_{i-1}$, that is, $T = T_1 T_2 \cdots T_{i-1} \hat{T}_i$ for some text suffix $\hat{T}_i$. The LZ77 dictionary is defined as the set of strings of the form $w\alpha$ where $\alpha \in \Sigma$ and w is a substring of T starting in the already parsed portion of T. The next word T_i is defined as the longest prefix of $\hat{T}_i$ which is a dictionary word. For example, for $T = aabbaaabaabaabba$ the LZ77 parsing is: a, ab, ba, $aaba$, $abaabb$, a. Note that, in some sense, $T_5 = abaabb$ is defined in terms of itself: it is a copy of the dictionary word $w\alpha$ with w starting at the second a of T_4 and extending into T_5! It is easy to see that all words in the parsing are distinct, with the possible exception of the last one (in the example the word a), and that the number of words in the LZ77 parsing is smaller than in the LZ78 parsing. If $T_i = w\alpha$ with $\alpha \in \Sigma$, the codeword for T_i is the triplet (s_i, ℓ_i, α) where s_i is the distance from the start of T_i to the last occurrence of w in $T_1 T_2 \cdots T_{i-1}$, and $\ell_i = |w|$.

Entropy Bounds

The performance of dictionary-based compressors has been extensively investigated since their introduction. In [15] it is shown that LZ77 is optimal for a certain family of sources, and in [16] it is shown that LZ78 achieves asymptotically the best compression ratio attainable by a finite-state compressor. This implies that, when the input string is generated by an ergodic source, the compression ratio achieved by LZ78 approaches the entropy of the source. More recent work has established similar results for other Ziv–Lempel compressors and has investigated the rate of convergence of the compression ratio to the entropy of the source (see [14] and references therein).

It is possible to prove compression bounds without probabilistic assumptions on the input, using the notion of *empirical entropy*. For any string T, the order k empirical entropy $H_k(T)$ is the maximum compression one can achieve using a uniquely decodable code in which the codeword for each character may depend on the k characters immediately preceding it [6]. The following lemma is a useful tool for establishing upper bounds on the compression ratio of dictionary-based algorithms which hold pointwise on every string T.

Lemma 1 ([6, Lemma 2.3]) *Let $T = T_1 T_2 \cdots T_d$ be a parsing of T such that each word T_i appears at most M times. Then, for any $k \ge 0$*

$$d \log d \le |T| H_k(T) + d \log(|T|/d) + d \log M + \Theta(kd + d),$$

where $H_k(T)$ is the k-th order empirical entropy of T. □

Consider, for example, the algorithm LZ78. It parses the input T into t distinct words (ignoring the last word in the parsing) and produces an output bounded by $t \log t + t \log |\Sigma| + \Theta(t)$ bits. Using Lemma 1 and the fact that $t = O(|T| / \log T)$, one can prove that LZ78′s output is at most $|T| H_k(T) + o(|T|)$ bits. Note that the bound holds for any $k \ge 0$: this means that LZ78 is essentially "as powerful" as any compressor that encodes the next character on the basis of a finite context.

Algorithmic Issues

One of the reasons for the popularity of dictionary-based compressors is that they admit linear-time, space-efficient implementations. These implementations sometimes require non-trivial data structures: the reader is referred to [12] and references therein for further reading on this topic.

Greedy vs. Non-Greedy Parsing

Both LZ78 and LZ77 use a greedy parsing strategy in the sense that, at each step, they select the longest prefix of the unparsed portion which is in the dictionary. It is easy to see that for LZ77 the greedy strategy yields an optimal parsing; that is, a parsing with the minimum number of words. Conversely, greedy parsing is not optimal for LZ78: for any sufficiently large integer m there exists a string that can be parsed to $O(m)$ words and that the greedy strategy parses in $\Omega(m^{3/2})$ words. In [9] the authors describe an efficient algorithm for computing an optimal parsing for the LZ78 dictionary and, indeed, for any dictionary with the prefix-completeness property (a dictionary is prefix-complete if any prefix of a dictionary word is also in the dictionary). Interestingly, the algorithm in [9] is a one-step lookahead greedy algorithm: rather than choosing the longest possible prefix of the unparsed portion of the text, it chooses the prefix that results in the longest advancement in the *next* iteration.

Applications

The natural application field of dictionary-based compressors is lossless data compression (see, for example [13]). However, because of their deep mathematical properties, the Ziv–Lempel parsing rules have also found applications in other algorithmic domains.

Prefetching

Krishnan and Vitter [7] considered the problem of prefetching pages from disk into memory to anticipate users' requests. They combined LZ78 with a pre-existing prefetcher P_1 that is asymptotically at least as good as the best memoryless prefetcher, to obtain a new algorithm P that is asymptotically at least as good as the best finite-state prefetcher. LZ78′s dictionary can be viewed as a trie: parsing a string means starting at the root, descending one level for each character in the parsed string and, finally, adding a new leaf. Algorithm P runs LZ78 on the string of page requests as it receives them, and keeps a copy of the simple prefetcher P_1 for each node in the trie; at each step, P prefetches the page requested by the copy of P_1 associated with the node LZ78 is currently visiting.

String Alignment

Crochemore, Landau and Ziv-Ukelson [4] applied LZ78 to the problem of sequence alignment, i. e., finding the cheapest sequence of character insertions, deletions and substitutions that transforms one string T into another T' (the cost of an operation may depend on the character or characters involved). Assume, for simplicity, that $|T| = |T'| = n$. In 1980 Masek and Paterson proposed an $O(n^2/\log n)$-time algorithm with the restriction that the costs be rational; Crochemore et al.'s algorithm allows real-valued costs, has the same asymptotic cost in the worst case, and is asymptotically faster for compressible texts.

The idea behind both algorithms is to break into blocks the matrix $A[1\dots n, 1\dots n]$ used by the obvious $O(n^2)$-time dynamic programming algorithm. Masek and Paterson break it into uniform-sized blocks, whereas Crochemore et al. break it according to the LZ78 parsing of T and T'. The rationale is that, by the nature of LZ78 parsing, whenever they come to solve a block $A[i\dots i', j\dots j']$, they can solve it in $O(i' - i + j' - j)$ time because they have already solved blocks identical to $A[i\dots i' - 1, j\dots j']$ and $A[i\dots i', j\dots j' - 1]$ [8]. Lifshits, Mozes, Weimann and Ziv-Ukelson [8 recently used a similar approach to speed up the decoding and training of hidden Markov models.

Compressed Full-Text Indexing

Given a text T, the problem of compressed full-text indexing is defined as the task of building an index for T that takes space proportional to the entropy of T and that supports the efficient retrieval of the occurrences of any pattern P in T. In [10] Navarro proposed a compressed full-text index based on the LZ78 dictionary. The basic idea is to keep two copies of the dictionary as tries: one storing the dictionary words, the other storing their reversal. The rationale behind this scheme is the following. Since any non-empty prefix of a dictionary word is also in the dictionary, if the sought pattern P occurs within a dictionary word, then P is a suffix of some word and easy to find in the second dictionary. If P overlaps two words, then some prefix of P is a suffix of the first word—and easy to find in the second dictionary—and the remainder of P is a prefix of the second word—and easy to find in the first dictionary. The case when P overlaps three or more words is a generalization of the case with two words. Recently, Arroyuelo et al. [1] improved the original data structure in [10]. For any text T, the improved index uses $(2+\epsilon)|T|H_k(T) + o(|T|\log|\Sigma|)$ bits of space, where $H_k(T)$ is the k-th order empirical entropy of T, and reports all *occ* occurrences of P in T in $O(|P|^2 \log|P| + (|P| + occ)\log|T|)$ time.

Independently of [10], in [5] the LZ78 parsing was used together with the Burrows-Wheeler compression algorithm to design the first full-text index that uses $o(|T| \log |T|)$ bits of space and reports the *occ* occurrences of P in T in $O(|P| + occ)$ time. If $T = T_1 T_2 \cdots T_d$ is the LZ78 parsing of T, in [5] the authors consider the string $T_\$ = T_1\$T_2\$\cdots\$T_d\$$ where \$ is a new character not belonging to Σ. The string $T_\$$ is then compressed using the Burrows-Wheeler transform. The \$'s play the role of anchor points: their positions in $T_\$$ are stored explicitly so that, to determine the position in T of any occurrence of P, it suffices to determine the position with respect to any of the \$'s. The properties of the LZ78 parsing ensure that the overhead of introducing the \$'s is small, but at the same time the way they are distributed within $T_\$$ guarantees the efficient location of the pattern occurrences.

Related to the problem of compressed full-text indexing is the compressed matching problem in which text and pattern are given together (so the former cannot be preprocessed). Here the task consists in performing string matching in a compressed text without decompressing it. For dictionary-based compressors this problem was first raised in 1994 by A. Amir, G. Benson, and M. Farach, and has received considerable attention since then. The reader is referred to [11] for a recent review of the many theoretical and practical results obtained on this topic.

Substring Compression Problems

Substring compression problems involve preprocessing T to be able to efficiently answer queries about compressing substrings: e. g., how compressible is a given substring s in T? what is s's compressed representation? or, what is the least compressible substring of a given length ℓ? These are important problems in bioinformatics because the compressibility of a DNA sequence may give hints as to its function, and because some clustering algorithms use compressibility to measure similarity. The solutions to these problems are often trivial for simple compressors, such as Huffman coding or run-length encoding, but they are open for more powerful algorithms, such as dictionary-based compressors, BWT compressors, and PPM compressors. Recently, Cormode and Muthukrishnan [3] gave some preliminary solutions for LZ77. For any string s, let $C(s)$ denote the number of words in the LZ77-parsing of s, and let LZ77(s) denote the LZ77-compressed representation of s. In [3] the authors show that, with $O(|T| \operatorname{polylog}(|T|))$ time preprocessing, for any substring s of T they can: *a*) compute LZ77(s) in $O(C(s) \log |T| \log\log |T|)$ time, *b*) compute an approximation of $C(s)$ within a factor $O(\log |T| \log^* |T|)$ in $O(1)$ time, *c*) find a substring of length ℓ that is close to being the least compressible in $O(|T|\ell/\log \ell)$ time. These bounds also apply to general versions of these problems, in which queries specify another substring t in T as context and ask about compressing substrings when LZ77 starts with a dictionary already containing the words in the LZ77 parsing of t.

Grammar Generation

Charikar et al. [2] considered LZ78 as an approximation algorithm for the NP-hard problem of finding the smallest context-free grammar that generates only the string T. The LZ78 parsing of T can be viewed as a context-free grammar in which for each dictionary word $T_i = T_j\alpha$ there is a production $X_i \to X_j\alpha$. For example, for $T = aabbaaabaabaabba$ the LZ78 parsing is: a, ab, b, aa, aba, $abaa$, bb, a, and the corresponding grammar is: $S \to X_1 \ldots X_7 X_1, X_1 \to a, X_2 \to X_1 b, X_3 \to b, X_4 \to X_1 a, X_5 \to X_2 a, X_6 \to X_5 a, X_7 \to X_3 b$. Charikar et al. showed LZ78's approximation ratio is in $O((|T|/\log |T|)^{2/3}) \cap \Omega(|T|^{2/3} \log |T|)$; i. e., the grammar it produces has size at most $f(|T|) \cdot m^*$, where $f(|T|)$ is a function in this intersection and m^* is the size of the smallest grammar. They also showed m^* is at least the number of words output by LZ77 on T, and used LZ77 as the basis of a new algorithm with approximation ratio $O(\log(|T|/m^*))$.

URL to Code

The source code of the gzip tool (based on LZ77) is available at the page http://www.gzip.org/. An LZ77-based compression library zlib is available from http://www.zlib.net/. A more recent, and more efficient, dictionary-based compressor is LZMA (Lempel–Ziv Markov chain Algorithm), whose source code is available from http://www.7-zip.org/sdk.html.

Cross References

- ▶ Arithmetic Coding for Data Compression
- ▶ Boosting Textual Compression
- ▶ Burrows–Wheeler Transform
- ▶ Compressed Text Indexing

Recommended Reading

1. Arroyuelo, D., Navarro, G., Sadakane, K.: Reducing the space requirement of LZ-index. In: Proc. 17th Combinatorial Pattern Matching conference (CPM), LNCS no. 4009, pp. 318–329, Springer (2006)
2. Charikar, M., Lehman, E., Liu, D., Panigraphy, R., Prabhakaran, M., Sahai, A., Shelat, A.: The smallest grammar problem. IEEE Trans. Inf. Theor. **51**, 2554–2576 (2005)

3. Cormode, G., Muthukrishnan, S.: Substring compression problems. In: Proc. 16th ACM-SIAM Symposium on Discrete Algorithms (SODA '05), pp. 321–330 (2005)
4. Crochemore, M., Landau, G., Ziv-Ukelson, M.: A subquadratic sequence alignment algorithm for unrestricted scoring matrices. SIAM J. Comput. **32**, 1654–1673 (2003)
5. Ferragina, P., Manzini, G.: Indexing compressed text. J. ACM **52**, 552–581 (2005)
6. Kosaraju, R., Manzini, G.: Compression of low entropy strings with Lempel–Ziv algorithms. SIAM J. Comput. **29**, 893–911 (1999)
7. Krishnan, P., Vitter, J.: Optimal prediction for prefetching in the worst case. SIAM J. Comput. **27**, 1617–1636 (1998)
8. Lifshits, Y., Mozes, S., Weimann, O., Ziv-Ukelson, M.: Speeding up HMM decoding and training by exploiting sequence repetitions. Algorithmica to appear doi:10.1007/s00453-007-9128-0
9. Matias, Y., Şahinalp, C.: On the optimality of parsing in dynamic dictionary based data compression. In: Proceedings 10th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '99), pp. 943–944 (1999)
10. Navarro, G.: Indexing text using the Ziv–Lempel trie. J. Discret. Algorithms **2**, 87–114 (2004)
11. Navarro, G., Tarhio, J.: LZgrep: A Boyer-Moore string matching tool for Ziv–Lempel compressed text. Softw. Pract. Exp. **35**, 1107–1130 (2005)
12. Şahinalp, C., Rajpoot, N.: Dictionary-based data compression: An algorithmic perspective. In: Sayood, K. (ed.) Lossless Compression Handbook, pp. 153–167. Academic Press, USA (2003)
13. Salomon, D.: Data Compression: the Complete Reference, 4th edn. Springer, London (2007)
14. Savari, S.: Redundancy of the Lempel–Ziv incremental parsing rule. IEEE Trans. Inf. Theor. **43**, 9–21 (1997)
15. Ziv, J., Lempel, A.: A universal algorithm for sequential data compression. IEEE Trans. Inf. Theor. **23**, 337–343 (1977)
16. Ziv, J., Lempel, A.: Compression of individual sequences via variable-length coding. IEEE Trans. Inf. Theor. **24**, 530–536 (1978)

Dictionary Matching and Indexing (Exact and with Errors)

2004; Cole, Gottlieb, Lewenstein

MOSHE LEWENSTEIN
Department of Computer Science, Bar Ilan University, Ramat-Gan, Israel

Keywords and Synonyms

Approximate dictionary matching; Approximate text indexing

Problem Definition

Indexing and *dictionary matching* are generalized models of pattern matching. These models have attained importance with the explosive growth of multimedia, digital libraries, and the Internet.

1. Text Indexing: In text indexing one desires to preprocess a text t, of length n, and to answer where subsequent queries p, of length m, appear in the text t.
2. Dictionary Matching: In dictionary matching one is given a dictionary D of strings $p_1, \dots, p_d$ to be preprocessed. Subsequent queries provide a query string t, of length n, and ask for each location in t at which patterns of the dictionary appear.

Key Results

Text Indexing

The *indexing* problem assumes a large text that is to be preprocessed in a way that will allow the following efficient future queries. Given a query pattern, one wants to find all text locations that match the pattern in time proportional to the *pattern length and to the number of occurrences*.

To solve the indexing problem, Weiner [14] invented the *suffix tree* data structure (originally called a *position tree*), which can be constructed in linear time, and subsequent queries of length m are answered in time $O(m \log|\Sigma| + tocc)$, where $tocc$ is the number of pattern occurrences in the text.

Weiner's suffix tree in effect solved the indexing problem for exact matching of fixed texts. The construction was simplified by the algorithms of McCreight and, later, Chen and Seiferas. Ukkonen presented an online construction of the suffix tree. Farach presented a linear time construction for large alphabets (specifically, when the alphabet is $\{1, \dots, n^c\}$, where n is the text size and c is some fixed constant). All results, besides the latter, work by handling one suffix at a time. The latter algorithm uses a divide and conquer approach, dividing the suffixes to be sorted to even-position suffixes and odd-position suffixes. See the entry on Suffix Tree Construction for full details. The standard query time for finding a pattern p in a suffix tree is $O(m \log|\Sigma|)$. By slightly adjusting the suffix tree one can obtain a query time of $O(m + \log n)$, see [12].

Another popular data structure for indexing is suffix arrays. Suffix arrays were introduced by Manber and Myers. Others proposed linear time constructions for linearly bounded alphabets. All three extend the divide and conquer approach presented by Farach. The construction in [11] is especially elegant and significantly simplifies the divide and conquer approach, by dividing the suffix set into three groups instead of two. See the entry on Suffix Array Construction for full details. The query time for suffix arrays is $O(m + \log n)$ achievable by embedding additional lcp (longest common prefix) information into the data structure. See [11] for reference to other solutions. *Suffix Trays* were introduced in [5] as a merge between suffix trees and suffix arrays. The construction time of suffix trays is the same as for suffix trees and suffix arrays. The query time is $O(m + \log|\Sigma|)$.

Solutions for the indexing problem in *dynamic* texts, where insertions and deletions (of single characters or entire substrings) are allowed, appear in several papers, see [2] and references therein.

Dictionary Matching

Dictionary matching is, in some sense, the "inverse" of text indexing. The large body to be preprocessed is a set of patterns, called the *dictionary*. The queries are texts whose length is typically significantly smaller than the dictionary size. It is desired to find all (exact) occurrences of dictionary patterns in the text in time proportional to the *text length and to the number of occurrences*.

Aho and Corasick [1] suggested an automaton-based algorithm that preprocesses the dictionary in time $O(d)$ and answers a query in time $O(n + docc)$, where $docc$ is the number of occurrences of patterns within the text. Another approach to solving this problem is to use a generalized suffix tree. A *generalized suffix tree* is a suffix tree for a collection of strings. Dictionary matching is done for the dictionary of patterns. Specifically, a suffix tree is created for the generalized string $p_1\$_1 p_2\$_2 \cdots \$p_d\$_d$, where the $\$_i$'s are not in the alphabet. A randomized solution using a fingerprint scheme was proposed in [3]. In [7] a parallel work-optimal algorithm for dictionary matching was presented. Ferragina and Luccio [8] considered the problem in the external memory model and suggested a solution based upon the String B-tree data structure along with the notion of a certificate for dictionary matching. Two Dimensional Dictionary Matching is another fascinating topic which appears as a separate entry. See also the entry on Multidimensional String Matching.

Dynamic Dictionary Matching: Here one allows insertion and deletion of patterns from the dictionary D. The first solution to the problem was a suffix tree-based method for solving the dynamic dictionary matching problem. Idury and Schäffer [10] showed that the failure function (function mapping from one longest matching prefix to the next longest matching prefix, see [1]) approach and basic scanning loop of the Aho–Corasick algorithm can be adapted to dynamic dictionary matching for improved initial dictionary preprocessing time. They also showed that faster search time can be achieved at the expense of slower dictionary update time.

A further improvement was later achieved by reducing the problem to maintaining a sequence of well-balanced parentheses under certain operations. In [13] an optimal method was achieved based on a labeling paradigm, where labels are given to, sometimes overlapping, substrings of different lengths. The running times are: $O(|D|)$ preprocessing time, $O(m)$ update time, and $O(n + docc)$ time for search. See [13] for other references.

Text Indexing and Dictionary Matching with Errors

In most real-life systems there is a need to allow errors. With the maturity of the solutions for *exact* indexing and *exact* dictionary matching, the quest for *approximate* solutions began. Two of the classical measures for approximating closeness of strings, Hamming distance and Edit distance, were the first natural measures to be considered.

Approximate Text Indexing: For approximate text indexing, given a distance k, one preprocesses a specified text t. The goal is to find all locations ℓ of t within distance k of the query p, i. e. for the Hamming distance all locations ℓ such that the length m substring of t beginning at that location can be made equal to p with at most k character substitutions. (An analogous statement applies for the edit distance.) For $k = 1$ [4] one can preprocess in time $O(n \log^2 n)$ and answer subsequent queries p in time $O(m\sqrt{\log n}\log\log n + occ)$. For small $k \geq 2$, the following naive solutions can be achieved. The first possible solution is to traverse a suffix tree checking all possible configurations of k, or less, mismatches in the pattern. However, while the preprocessing needed to build a suffix tree is cheap, the search is expensive, namely, $O(m^{k+1}|\Sigma|^k + occ)$. Another possible solution, for the Hamming distance measure only, leads to data structures of size approximately $O(n^{k+1})$ embedding all mismatch possibilities into the tree. This can be slightly improved by using the data structures for $k = 1$, which reduce the size to approximately $O(n^k)$.

Approximate Dictionary Matching: The goal is to preprocess the dictionary along with a threshold parameter k in order to support the following subsequent queries: Given a query text, seek all pairs of patterns (from the dictionary) and text locations which match within distance k. Here once again there are several algorithms for the case where $k = 1$ [4,9]. The best solution for this problem has query time $O(m \log\log n + occ)$; the data structure uses space $O(n \log n)$ and can be built in time $O(n \log n)$.

The solutions for $k = 1$ in both problems (Approximate Text Indexing and Approximate Dictionary Matching) are based on the following, elegant idea, presented in Indexing terminology. Say a pattern p matches a text t at location i with one error at location j of p (and at location $i + j - 1$ of t). Obviously, the $j - 1$-length prefix of p matches the aligned substring of t and so does the

$m - j - 1$ length suffix. If t and p are reversed then the $j - 1$-th length prefix of p becomes a $j - 1$-th length suffix of p^R (that is p reverse). Notice that there is a match with, at most one error, if (1) the suffix of p starting at location $j + 1$ matches the (prefix of the) suffix of t starting at location $i + j$ and (2) the suffix of p^R starting at location $m - j + 1$ (the reverse of the $j - 1$-th length prefix of p) matches the (prefix of the) suffix of t^R starting at location $m - i - j + 3$. So, the problem now becomes a search for locations j which satisfy the above. To do so, the above-mentioned solutions, naturally, use two suffix trees, one for the text and one for its reverse (with additional data structure tricks to answer the query fast). In dictionary matching the suffix trees are defined on the dictionary. The problem is that this solution does not carry over for $k \geq 2$. See the introduction of [6] for a full list of references.

Text Indexing and Dictionary Matching within (Small) Distance *k*

Cole et al. [6] proposed a new method that yields a unified solution for approximate text indexing, approximate dictionary matching, and other related problems. However, since the solution is somewhat involved it will be simpler to explain the ideas on the following problem. The desire is to index a text t to allow fast searching for all occurrences of a pattern containing, at most, k don't cares (don't cares are special characters which match all characters).

Once again, there are two possible, relatively straightforward, solutions to be elaborated. The first is to use a suffix tree, which is cheap to preprocess, but causes the search to be expensive, namely, $O(m|\Sigma|^k + occ)$ (if considering k mismatches this would increase to $O(m^{k+1}|\Sigma|^k + occ)$. To be more specific, imagine traversing a path in a suffix tree. Consider the point where a don't care is reached. If in the middle of an edge the only text suffixes (representing substrings) that can match the pattern with this don't care must also go through this edge. So simply continue traversing. However, if at a node, then all the paths leaving this node must be explored. This explains the mentioned time bound.

The second solution is to create a tree that contains all strings that are at Hamming distance k from a suffix. This allows fast search but leads to trees of size exponential in k, namely, $O(n^{k+1})$ size trees. To elaborate, the tree, called a *k-error-trie*, is constructed as follows. First, consider the case for one don't care, i. e. a *1-error-trie*, and then extend it. At any node v a don't care may need to be evaluated. Therefore, create a special subtree branching off this node that represents a don't care at this node. To understand this subtree, note that the subtree (of the suffix tree) rooted at v is actually a compressed trie of (some of the) suffixes of the text. Denote the collection of suffixes S_v. The first character of all these suffixes have to be removed (or, perhaps better imagined as a replacement with a don't care character). Each will be a new suffix of the text. Denote the new collection as S'_v. Now, create a new compressed trie of suffixes for S'_v, calling this new subtree an *error tree*. Do so for every v. The suffix tree along with its error trees is a *1-error-trie*. Turning to queries in the *1-error-trie*, when traversing the *1-error-trie*, do so with the suffix tree up till the don't care at node v. Move into the error tree at node v and continue the traversal of the pattern.

To create a *2-error-trie*, simply take each error tree and construct an error tree for each node within. A *(k+1)-error trie* is created recursively from a *k-error trie*. Clearly the *1-error trie* is of size $O(n^2)$, since any node u in the original suffix tree will appear in all the new subtrees of the *1-error trie* created for each of the nodes v which are ancestors of u. Likewise, the *k-error-trie* is of size $O(n^{k+1})$.

The method introduced in Cole et al. [6] uses the idea of the error trees to form a new data structure, which is called a *k-errata trie*. The *k-errata trie* will be much smaller than $O(n^{k+1})$. However, it comes at the cost of a somewhat slower search time. To understand the *k-errata tries* it is useful to first consider the *1-errata-tries* and to extend. The *1-errata-trie* is constructed as follows. The suffix tree is first decomposed with a centroid path decomposition (which is a decomposition of the nodes into paths, where all nodes along a path have their subtree sizes within a range 2^r and 2^{r+1}, for some integer r). Then, as before, error trees are created for each node v of the suffix tree with the following difference. Namely, consider the subtree, T_v, at node v and consider the edge (v, x) going from v to child x on the centroid path. T_v can be partitioned into two subtrees, $T_x \cup (v, x)$, and T'_v all the rest of T_v. An error tree is created for the suffixes in T'_v. The *1-errata-trie* is the suffix tree with all of its error trees. Likewise, a *(k+1)-errata trie* is created recursively from a *k-errata trie*. The contents of a *k-errata trie* should be viewed as a collection of error trees, k levels deep, where error trees at each level are constructed on the error trees of the previous level (at level 0 there is the original suffix tree). The following lemma helps in obtaining a bound on the size of the *k-errata trie*.

Lemma 1 *Let C be a centroid decomposition of a tree T. Let u be an arbitrary node of T and π be the path from the root to u. There are at most* $\log n$ *nodes v on π for which v and v's parent on π are on different centroid paths.*

The implication is that every node u in the original suffix tree will only appear in $\log n$ error trees of the *1-errata trie* because each ancestor v of u is on the path π from the root to u and only $\log n$ such nodes are on different centroid paths than their children (on π). Hence, u appears in only $\log^k n$ error trees in the *k-errata trie*. Therefore, the size of the *k-errata trie* is $O(n \log^k n)$. Creating the *k-errata tries* in $O(n \log^{k+1} n)$ can be done. To answer queries on a *k-errata trie*, given the pattern with (at most) k don't cares, the 0th level of the *k-errata trie*, i. e. the suffix tree, needs to be traversed. This is to be done until the first don't care, at location j, in the pattern is reached. If at node v in the 0th level of the *k-errata trie*, enter the (1st level) error tree hanging off of v and traverse this error tree from location $j + 2$ of the pattern (until the next don't care is met). However, the error tree hanging off of node v does not contain the subtree hanging off of v that is along the centroid path. Hence, continue traversing the pattern in the 0th level of the *k-errata trie*, starting along the edge on the centroid path leaving v (until the next don't care is met). The search is done recursively for k don't cares and, hence, yields an $O(2^k m)$ time search.

Recall that a solution for indexing text that supports queries of a pattern with k don't cares has been described. Unfortunately, when indexing to support k mismatch queries, not to mention k edit operation queries, the traversal down a *k-errata trie* can be very time consuming as frequent branching is required since an error may occur at any location of the pattern. To circumvent this problem search many error trees in parallel. In order to do so, the error trees have to be grouped together. This needs to be done carefully, see [6] for the full details. Moreover, edit distance needs even more careful handling. The time and space of the algorithms achieved in [6] are as follows:

Approximate Text Indexing: The data structure for mismatches uses space $O(n \log^k n)$, takes time $O(n \log^{k+1} n)$ to build, and answers queries in time $O((\log^k n) \log\log n + m + occ)$. For edit distance, the query time becomes $O((\log^k n) \log\log n + m + 3^k \cdot occ)$. It must be pointed out that this result is mostly effective for constant k.

Approximate Dictionary Matching: For k mismatches the data structure uses space $O(n + d \log^k d)$, is built in time $O(n + d \log^{k+1} d)$, and has a query time of $O((m + log^k d) \cdot \log\log n + occ)$. The bounds for edit distance are modified as in the indexing problem.

Applications

Approximate Indexing has a wide array of applications in signal processing, computational biology, and text retrieval among others. Approximate Dictionary Matching is important in digital libraries and text retrieval systems.

Cross References

- ▶ Compressed Text Indexing
- ▶ Indexed Approximate String Matching
- ▶ Multidimensional String Matching
- ▶ Sequential Multiple String Matching
- ▶ Suffix Array Construction
- ▶ Suffix Tree Construction in Hierarchical Memory
- ▶ Suffix Tree Construction in RAM
- ▶ Text Indexing
- ▶ Two-Dimensional Pattern Indexing

Recommended Reading

1. Aho, A.V., Corasick, M.J.: Efficient string matching. Commun. ACM **18**(6), 333–340 (1975)
2. Alstrup, S., Brodal, G.S., Rauhe, T.: Pattern matching in dynamic texts. In: Proc. of Symposium on Discrete Algorithms (SODA), 2000, pp. 819–828
3. Amir, A., Farach, M., Matias, Y.: Efficient randomized dictionary matching algorithms. In: Proc. of Symposium on Combinatorial Pattern Matching (CPM), 1992, pp. 259–272
4. Amir, A., Keselman, D., Landau, G.M., Lewenstein, N., Lewenstein, M., Rodeh, M.: Indexing and dictionary matching with one error. In: Proc. of Workshop on Algorithms and Data Structures (WADS), 1999, pp. 181–192
5. Cole, R., Kopelowitz, T., Lewenstein, M.: Suffix trays and suffix trists: Structures for faster text indexing. In: Proc. of International Colloquium on Automata, Languages and Programming (ICALP), 2006, pp. 358–369
6. Cole, R., Gottlieb, L., Lewenstein, M.: Dictionary matching and indexing with errors and don't cares. In: Proc. of the Symposium on Theory of Computing (STOC), 2004, pp. 91–100
7. Farach, M., Muthukrishnan, S.: Optimal parallel dictionary matching and compression. In: Symposium on Parallel Algorithms and Architecture (SPAA), 1995, pp. 244–253
8. Ferragina, P., Luccio, F.: Dynamic dictionary matching in external memory. Inf. Comput. **146**(2), 85–99 (1998)
9. Ferragina, P., Muthukrishnan, S., deBerg, M.: Multi-method dispatching: a geometric approach with applications to string matching. In: Proc. of the Symposium on the Theory of Computing (STOC), 1999, pp. 483–491
10. Idury, R.M., Schäffer, A.A.: Dynamic dictionary matching with failure functions. In: Proc. 3rd Annual Symposium on Combinatorial Pattern Matching, 1992, pp. 273–284
11. Karkkainen, J., Sanders, P., Burkhardt, S.: Linear work suffix array construction. J. ACM **53**(6), 918–936 (2006)
12. Mehlhorn, K.: Dynamic binary search. SIAM J. Comput. **8**(2), 175–198 (1979)
13. Sahinalp, S.C., Vishkin, U.: Efficient approximate and dynamic matching of patterns using a labeling paradigm. In: Proc. of the Foundations of Computer Science (FOCS), 1996, pp. 320–328
14. Weiner, P.: Linear pattern matching algorithm. In: Proc. of the Symposium on Switching and Automata Theory, 1973, pp. 1–11

Dilation

► Geometric Spanners
► Planar Geometric Spanners

Dilation of Geometric Networks

2005; Ebbers-Baumann, Grüne, Karpinski, Klein, Kutz, Knauer, Lingas

ROLF KLEIN
Institute for Computer Science, University of Bonn, Bonn, Germany

Keywords and Synonyms

Detour; Spanning ratio; Stretch factor

Problem Definition

Notations

Let $G = (V, E)$ be a plane geometric network, whose vertex set V is a finite set of point sites in $\mathbb{R}^2$, connected by an edge set E of non-crossing straight line segments with endpoints in V. For two points $p \neq q \in V$ let $\xi_G(p, q)$ denote a shortest path from p to q in G. Then

$$\sigma(p, q) := \frac{|\xi_G(p, q)|}{|pq|} \qquad (1)$$

is the detour one encounters when using network G, in order to get from p to q, instead of walking straight. Here, $|.|$ denotes the Euclidean length.

The *dilation* of G is defined by

$$\sigma(G) := \max_{p \neq q \in V} \sigma(p, q) \; . \qquad (2)$$

This value is also known as the spanning ratio or the stretch factor of G. It should, however, not be confused with the geometric dilation of a network, where the points on the edges are also being considered, in addition to the vertices.

Given a finite set S of points in the plane, one would like to find a plane geometric network $G = (V, E)$ whose dilation $\sigma(G)$ is as small as possible, such that S is contained in V. The value of

$$\Sigma(S) := \inf\{\, \sigma(G);\; G = (V, E) \text{ finite plane geometric network where } S \subset V \,\}$$

is called the *dilation of point set* S. The problem is in computing, or bounding, $\Sigma(S)$ for a given set S.

Related Work

If edge crossings were allowed one could use spanners whose stretch can be made arbitrarily close to 1; see the monographs by Eppstein [6] or Narasimhan and Smid [12]. Different types of triangulations of S are known to have their stretch factors bounded from above by small constants, among them the Delaunay triangulation of stretch ≤ 2.42; see Dobkin et al. [3], Keil and Gutwin [10], and Das and Joseph [2]. Eppstein [5] has characterized all triangulations T of dilation $\sigma(T) = 1$; these triangulations are shown in Fig. 1. Trivially, $\Sigma(S) = 1$ holds for each point set S contained in the vertex set of such a triangulation T.

Key Results

The previous remark's converse turns also out to be true.

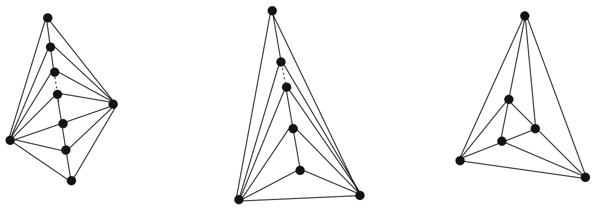
Theorem 1 ([11]) *If S is not contained in one of the vertex sets depicted in Fig. 1 then $\Sigma(S) > 1$.*

That is, if a point set S is not one of these special sets then each plane network including S in its vertex set has a dilation larger than some lower bound $1 + \eta(S)$. The proof of Theorem 1 uses the following density result. Suppose one connects each pair of points of S with a straight line segment. Let S' be the union of S and the resulting crossing points. Now the same construction is applied to S', and repeated. For the limit point set S^∞ the following theorem holds. It generalizes work by Hillar and Rhea [8] and by Ismailescu and Radoičić [9] on the intersections of lines.

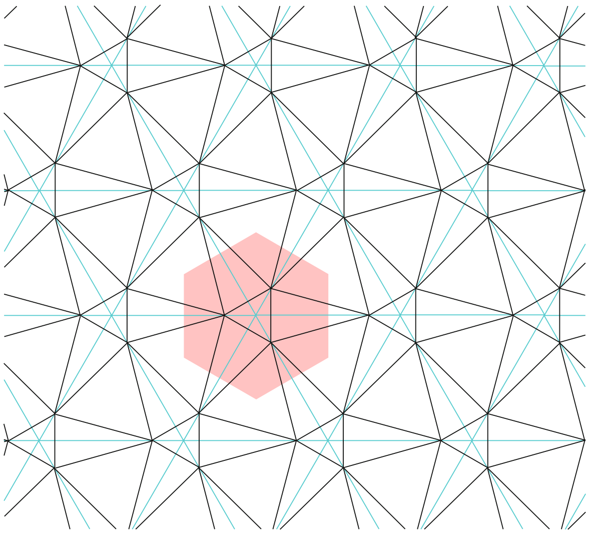
Theorem 2 ([11]) *If S is not contained in one of the vertex sets depicted in Fig. 1 then S^∞ lies dense in some polygonal part of the plane.*

For certain infinite structures can concrete lower bounds be proven.

Theorem 3 ([4]) *Let N be an infinite plane network all of whose faces have a diameter bounded from above by some constant. Then $\sigma(N) > 1.00156$ holds.*



Dilation of Geometric Networks, Figure 1
The triangulations of dilation 1



Dilation of Geometric Networks, Figure 2
A network of dilation ~ 1.1247

Theorem 4 ([4]) *Let C denote the (infinite) set of all points on a closed convex curve. Then $\Sigma(C) > 1.00157$ holds.*

Theorem 5 ([4]) *Given n families F_i, $2 \leq i \leq n$, each consisting of infinitely many equidistant parallel lines. Suppose that these families are in general position. Then their intersection graph G is of dilation at least $2/\sqrt{3}$.*

The proof of Theorem 5 makes use of Kronecker's theorem on simultaneous approximation. The bound is attained by the packing of equiangular triangles.

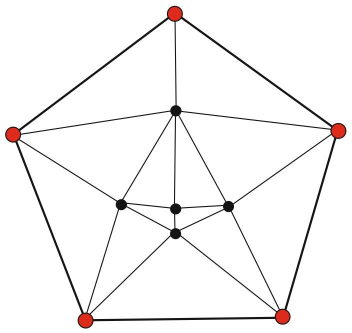
Finally, there is a general upper bound to the dilation of finite point sets.

Theorem 6 ([4]) *Each finite point set S is of dilation $\Sigma(S) < 1.1247$.*

To prove this upper bound one can embed any given finite point set S in the vertex set of a scaled, and slightly deformed, finite part of the network depicted in Fig. 2. It results from a packing of equilateral triangles by replacing each vertex with a small triangle, and by connecting neighboring triangles as indicated.

Applications

A typical university campus contains facilities like lecture halls, dorms, library, mensa, and supermarkets, which are connected by some path system. Students in a hurry are tempted to walk straight across the lawn, if the shortcut seems worth it. After a while, this causes new paths to appear. Since their intersections are frequented by many people, they attract coffee shops or other new facilities. Now,



Dilation of Geometric Networks, Figure 3
The best known embedding for S_5

people will walk across the lawn to get quickly to a coffee shop, and so on.

D. Eppstein [5] has asked what happens to the lawn if this process continues. The above results show that (1) part of the lawn will be completely destroyed, and (2) the temptation to walk across the lawn cannot, in general, be made arbitrarily small by a clever path design.

Open Problems

For practical applications, upper bounds to the weight (= total edge length) of a geometric network would be valuable, in addition to upper dilation bounds. Some theoretical questions require further investigation, too. Is $\Sigma(S)$ always attained by a finite network? How to compute, or approximate, $\Sigma(S)$ for a given finite set S? Even for a set as simple as S_5, the corners of a regular 5-gon, is the dilation unknown. The smallest dilation value known, for a triangulation containing S_5 among its vertices, equals 1.0204; see Fig. 3. Finally, what is the precise value of $\sup\{\Sigma(S); S \text{ finite}\}$?

Cross References

► Geometric Dilation of Geometric Networks

Recommended Reading

1. Aronov, B., de Berg, M., Cheong, O., Gudmundsson, J., Haverkort, H., Vigneron, A.: Sparse Geometric Graphs with Small Dilation. 16th International Symposium ISAAC 2005, Sanya. In: Deng, X., Du, D. (eds.) Algorithms and Computation, Proceedings. LNCS, vol. 3827, pp. 50–59. Springer, Berlin (2005)
2. Das, G., Joseph, D.: Which Triangulations Approximate the Complete Graph? In: Proc. Int. Symp. Optimal Algorithms. LNCS 401, pp. 168–192. Springer, Berlin (1989)
3. Dobkin, D.P., Friedman, S.J., Supowit, K.J.: Delaunay Graphs Are Almost as Good as Complete Graphs. Discret. Comput. Geom. **5**, 399–407 (1990)

4. Ebbers-Baumann, A., Gruene, A., Karpinski, M., Klein, R., Knauer, C., Lingas, A.: Embedding Point Sets into Plane Graphs of Small Dilation. Int. J. Comput. Geom. Appl. **17**(3), 201–230 (2007)
5. Eppstein, D.: The Geometry Junkyard. http://www.ics.uci.edu/~eppstein/junkyard/dilation-free/
6. Eppstein, D.: Spanning Trees and Spanners. In: Sack, J.-R., Urrutia, J. (eds.) Handbook of Computational Geometry, pp. 425–461. Elsevier, Amsterdam (1999)
7. Eppstein, D., Wortman, K.A.: Minimum Dilation Stars. In: Proc. 21st ACM Symp. Comp. Geom. (SoCG), Pisa, 2005, pp. 321–326
8. Hillar, C.J., Rhea, D.L. A Result about the Density of Iterated Line Intersections. Comput. Geom.: Theory Appl. **33**(3), 106–114 (2006)
9. Ismailescu, D., Radoičić, R.: A Dense Planar Point Set from Iterated Line Intersections. Comput. Geom. Theory Appl. **27**(3), 257–267 (2004)
10. Keil, J.M., Gutwin, C.A.: The Delaunay Triangulation Closely Approximates the Complete Euclidean Graph. Discret. Comput. Geom. **7**, 13–28 (1992)
11. Klein, R., Kutz, M.: The Density of Iterated Plane Intersection Graphs and a Gap Result for Triangulations of Finite Point Sets. In: Proc. 22nd ACM Symp. Comp. Geom. (SoCG), Sedona (AZ), 2006, pp. 264–272
12. Narasimhan, G., Smid, M.: Geometric Spanner Networks. Cambridge University Press (2007)

Directed Perfect Phylogeny (Binary Characters)
1991; Gusfield

JESPER JANSSON
Ochanomizu University, Tokyo, Japan

Keywords and Synonyms

Directed binary character compatibility

Problem Definition

Let $S = \{s_1, s_2, \dots, s_n\}$ be a set of elements called *objects*, and let $C = \{c_1, c_2, \dots, c_m\}$ be a set of functions from S to $\{0, 1\}$ called *characters*. For each object $s_i \in S$ and character $c_j \in C$, it is said that s_i *has* c_j if $c_j(s_i) = 1$ or that s_i *does not have* c_j if $c_j(s_i) = 0$, respectively (in this sense, characters are *binary*). Then the set S and its relation to C can be naturally represented by a matrix M of size $(n \times m)$ satisfying $M[i, j] = c_j(s_i)$ for every $i \in \{1, 2, \dots, n\}$ and $j \in \{1, 2, \dots, m\}$. Such a matrix M is called a *binary character state matrix.*

Next, for each $s_i \in S$, define the set $C_{s_i} = \{c_j \in C\colon s_i \text{ has } c_j\}$. A *phylogeny for S* is a tree whose leaves are bijectively labeled by S, and a *directed perfect phylogeny for (S, C)* (if one exists) is a rooted phylogeny T for S in which each $c_j \in C$ is associated with exactly one edge of T in such a way that for any $s_i \in S$, the set of all characters associated with the edges on the path in T from the root to leaf s_i is equal to C_{s_i}. See Figs. 1 and 2 for two examples.

Now, define the following problem.

Problem 1 (The Directed Perfect Phylogeny Problem for Binary Characters)
INPUT: *A binary character state matrix M for some S and C.*
OUTPUT: *A directed perfect phylogeny for (S, C), if one exists; otherwise, null.*

Key Results

For the presentation below, for each $c_j \in C$, define a set $S_{c_j} = \{s_i \in S : s_i \text{ has } c_j\}$. The next lemma is the key to solving The Directed Perfect Phylogeny Problem for Binary Characters efficiently. It was first proved by Estabrook, Johnson, and McMorris [2,3], and is also known in the literature as *the pairwise compatibility theorem*. A constructive proof of the lemma can be found in, e. g., [7,11].

Lemma 1 ([2,3]) *There exists a directed perfect phylogeny for (S, C) if and only if for all $c_j, c_k \in C$ it holds that $S_{c_j} \cap S_{c_k} = \emptyset$, $S_{c_j} \subseteq S_{c_k}$, or $S_{c_k} \subseteq S_{c_j}$.*

Using Lemma 1, it is straightforward to construct a top-down algorithm for the problem that runs in $O(nm^2)$ time. However, a faster algorithm is possible. Gusfield [6] observed that after sorting the columns of M in non-increasing order all duplicate copies of a column appear in a consecutive block of columns and column j is to the right of column k if S_{c_j} is a proper subset of S_{c_k}, and exploited this fact together with Lemma 1 to obtain the following result:

Theorem 2 ([6]) *The Directed Perfect Phylogeny Problem for Binary Characters can be solved in $O(nm)$ time.*

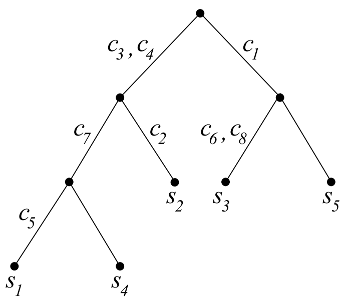
For a detailed description of the original algorithm and a proof of its correctness, see [6] or [11]. A conceptually simplified version of the algorithm based on keyword trees can be found in [7]. Gusfield [6] also gave an adversary argument to prove a corresponding lower bound of $\Omega(nm)$ on the running time, showing that his algorithm is time optimal:

Theorem 3 ([6]) *Any algorithm that decides if a given binary character state matrix M admits a directed perfect phylogeny must, in the worst case, examine all entries of M.*

Agarwala, Fernández-Baca, and Slutzki [1] noted that the input binary character state matrix is often sparse, i. e., in general, most of the objects will not have most of the characters. In addition, they noted that for the sparse case, it is more efficient to represent the input (S, C) by all the sets S_{c_j} for $j \in \{1, 2, \dots, m\}$, where each set S_{c_j} is defined

$$\begin{array}{c|cccccccc} M & c_1 & c_2 & c_3 & c_4 & c_5 & c_6 & c_7 & c_8 \\ \hline s_1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ s_2 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ s_3 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ s_4 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ s_5 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

a



b

Directed Perfect Phylogeny (Binary Characters), Figure 1
a A (5 × 8)-binary character state matrix *M*. b A directed perfect phylogeny for (*S*,*C*)

$$\begin{array}{c|cc} M & c_1 & c_2 \\ \hline s_1 & 1 & 0 \\ s_2 & 1 & 1 \\ s_3 & 0 & 1 \end{array}$$

Directed Perfect Phylogeny (Binary Characters), Figure 2
This binary character state matrix admits no directed perfect phylogeny

as above and each S_{c_j} is specified as a linked list, than by using a binary character state matrix. Agarwala et al. [1] proved that with this alternative representation of S and C, the algorithm of Gusfield can be modified to run in time proportional to the total number of 1's in the corresponding binary character state matrix[1]:

Theorem 4 ([1]) *The variant of The Directed Perfect Phylogeny Problem for Binary Characters in which the input is given as linked lists representing all the sets S_{c_j} for $j \in \{1, 2, \dots, m\}$ can be solved in $O(h)$ time, where $h = \sum_{j=1}^{m} |S_{c_j}|$.*

For a description of the algorithm, refer to [1] or [5].

Applications

Directed perfect phylogenies for binary characters are used to describe the evolutionary history for a set of objects that share some observable traits and that have evolved from a "blank" ancestral object which has none of the traits. Intuitively, the root of a directed perfect phylogeny corresponds to the blank ancestral object and each directed edge $e = (u, v)$ corresponds to an evolutionary event in which the hypothesized ancestor represented by u gains the characters associated with e, transforming it into the hypothesized ancestor or object represented by v. It is assumed that each character can emerge once only during the evolutionary history and is never lost after it has been gained[2], so a leaf s_i is a descendant of the edge associated with a character c_j if and only if s_i has c_j.

Binary characters are commonly used by biologists and linguists. Traditionally, morphological traits or directly observable features of species were employed by biologists as binary characters, and recently, binary characters based on genomic information such as substrings in DNA or protein sequences, protein regulation data, and shared gaps in a given multiple alignment have become more and more prevalent. Section 17.3.2 in [7] mentions several examples where phylogenetic trees have been successfully constructed based on such types of binary character data. In the context of reconstructing the evolutionary history of natural languages, linguists often use phonological and morphological characters with just two states [9].

The Directed Perfect Phylogeny Problem for Binary Characters is closely related to *The Perfect Phylogeny Problem*, a fundamental problem in computational evolutionary biology and phylogenetic reconstruction [4,5,11]. This problem (also described in more detail in entry ▶ Perfect Phylogeny (Bounded Number of States)) introduces non-binary characters so that each character $c_j \in C$ has a set of allowed states $\{0, 1, \dots, r_j - 1\}$ for some integer r_j, and for each $s_i \in S$, character c_j is in one of its allowed states. Generalizing the notation used above, define the set $S_{c_j,\alpha}$ for every $\alpha \in \{0, 1, \dots, r_j - 1\}$ by $S_{c_j,\alpha} = \{s_i \in S : \text{the state of } s_i \text{ on } c_j \text{ is } \alpha\}$. Then, the objective of *The Perfect Phylogeny Problem* is to construct (if possible) an *unrooted* phylogeny T for S such that the following holds: for each $c_j \in C$ and distinct states α, β of c_j,

[1]Note that Theorem 4 does not contradict Theorem 3; in fact, Gusfield's lower bound argument considers an input matrix consisting mostly of 1's.

[2]When this requirement is too strict, one can relax it to permit errors; for example, let characters be associated with more than one edge in the phylogeny (i. e., allow each character to emerge many times) but minimize the total number of associations (*Camin–Sokal optimization*), or keep the requirement that each character emerges only once but allow it to be lost multiple times (*Dollo parsimony*) [4,5]

the minimal subtree of T that connects $S_{c_j,\alpha}$ and the minimal subtree of T that connects $S_{c_j,\beta}$ are vertex-disjoint. McMorris [10] showed that the special case with $r_j = 2$ for all $c_j \in C$ can be reduced to The Directed Perfect Phylogeny Problem for Binary Characters in $O(nm)$ time (for each $c_j \in C$, if the number of 1's in column j of M is greater than the number of 0's then set entry $M[i, j]$ to $1 - M[i, j]$ for all $i \in \{1, 2, \ldots, n\}$). Therefore, another application of Gusfield's algorithm [6] is as a subroutine for solving The Perfect Phylogeny Problem when $r_j = 2$ for all $c_j \in C$ in $O(nm)$ time. Even more generally, The Perfect Phylogeny Problem for directed as well as undirected *cladistic* characters can be solved in polynomial time by a similar reduction to The Directed Perfect Phylogeny Problem for Binary Characters (see [5]).

In addition to the above, it is possible to apply Gusfield's algorithm to determine whether two given trees describe compatible evolutionary history, and if so, merge them into a single tree so that no branching information is lost (see [6] for details). Finally, Gusfield's algorithm has also been used by Hanisch, Zimmer, and Lengauer [8] to implement a particular operation on documents defined in their Protein Markup Language (ProML) specification.

Cross References

► Perfect Phylogeny (Bounded Number of States)
► Perfect Phylogeny Haplotyping

Acknowledgments

Supported in part by Kyushu University, JSPS (Japan Society for the Promotion of Science), and INRIA Lille - Nord Europe.

Recommended Reading

1. Agarwala, R., Fernández-Baca, D., Slutzki, G.: Fast algorithms for inferring evolutionary trees. J. Comput. Biol. **2**, 397–407 (1995)
2. Estabrook, G.F., Johnson, C.S., Jr., McMorris, F.R.: An algebraic analysis of cladistic characters. Discret. Math. **16**, 141–147 (1976)
3. Estabrook, G.F., Johnson, C.S., Jr., McMorris, F.R.: A mathematical foundation for the analysis of cladistic character compatibility. Math. Biosci. **29**, 181–187 (1976)
4. Felsenstein, J.: Inferring Phylogenies. Sinauer Associates, Inc., Sunderland (2004)
5. Fernández-Baca, D.: The Perfect Phylogeny Problem. In: Cheng, X., Du, D.-Z. (eds.) Steiner Trees in Industry, pp. 203–234. Kluwer Academic Publishers, Dordrecht (2001)
6. Gusfield, D.M.: Efficient algorithms for inferring evolutionary trees. Networks **21**, 19–28 (1991)
7. Gusfield, D.M.: Algorithms on Strings, Trees, and Sequences. Cambridge University Press, New York (1997)
8. Hanisch, D., Zimmer, R., Lengauer, T.: ProML – the Protein Markup Language for specification of protein sequences, structures and families. In: Silico Biol. **2**, 0029 (2002). http://www.bioinfo.de/isb/2002/02/0029/
9. Kanj, I.A., Nakhleh, L., Xia, G.: Reconstructing evolution of natural languages: Complexity and parametrized algorithms. In: Proceedings of the 12th Annual International Computing and Combinatorics Conference (COCOON 2006). Lecture Notes in Computer Science, vol. 4112, pp. 299–308. Springer, Berlin (2006)
10. McMorris, F.R.: On the compatibility of binary qualitative taxonomic characters. Bull. Math. Biol. **39**, 133–138 (1977)
11. Setubal, J.C., Meidanis, J.: Introduction to Computational Molecular Biology. PWS Publishing Company, Boston (1997)

Direct Routing Algorithms

2006; Busch, Magdon-Ismail, Mavronicolas, Spirakis

COSTAS BUSCH
Department of Computer Science,
Lousiana State University, Baton Rouge, LA, USA

Keywords and Synonyms

Hot-potato routing; Bufferless packet switching; Collision-free packet scheduling

Problem Definition

The performance of a communication network is affected by the *packet collisions* which occur when two or more packets appear simultaneously in the same network node (router) and all these packets wish to follow the same outgoing link from the node. Since network links have limited available bandwidth, the collided packets wait on buffers until the collisions are resolved. Collisions cause delays in the packet delivery time and also contribute to the network performance degradation.

Direct routing is a packet delivery method which avoids packet collisions in the network. In direct routing, after a packet is injected into the network it follows a path to its destination without colliding with other packets, and thus without delays due to buffering, until the packet is absorbed at its destination node. The only delay that a packet experiences is at the source node while it waits to be injected into the network.

In order to formulate the direct routing problem, the network is modeled as a graph where all the network nodes are synchronized with a common time clock. Network links are bidirectional, and at each time step any link can be crossed by at most two packets, one packet in each direction. Given a set of packets, the *routing time* is defined to be the time duration between the first packet injection and the last packet absorbtion.

Consider a set of N packets, where each packet has its own source and destination node. In the *direct rout-*

ing problem, the goal is first to find a set of paths for the packets in the network, and second, to find appropriate injection times for the packets, so that if the packets are injected at the prescribed times and follow their paths they will be delivered to their destinations without collisions. The *direct scheduling problem* is a variation of the above problem, where the paths for the packets are given a priori, and the only task is to compute the injection times for the packets.

A *direct routing algorithm* solves the direct routing problem (similarly, a *direct scheduling algorithm* solves the direct scheduling problem). The objective of any direct algorithm is to minimize the routing time for the packets. Typically, direct algorithms are *offline*, that is, the paths and the injection schedule are computed ahead of time, before the packets are injected into the network, since the involved computation requires knowledge about all packets in order to guarantee the absence of collisions between them.

Key Results

Busch, Magdon-Ismail, Mavronicolas, and Spirakis, present in [6] a comprehensive study of direct algorithms. They study several aspects of direct routing such as the computational complexity of direct problems and also the design of efficient direct algorithms. The main results of their work are described below.

Hardness of Direct Routing

It is shown in [Sect. 4 in 6] that the optimal direct scheduling problem, where the paths are given and the objective is to compute an optimal injection schedule (that minimizes the routing time) is an NP-complete problem. This result is obtained with a reduction from vertex coloring, where vertex coloring problems are transformed to appropriate direct scheduling problems in a 2-dimensional grid. In addition, it is shown in [6] that approximations to the direct scheduling problem are as hard to obtain as approximations to vertex coloring. A natural question is what kinds of approximations can be obtained in polynomial time. This question is explored in [6] for general and specific kinds of graphs, as described below.

Direct Routing in General Graphs

A direct algorithm is given in [Section 3 in 6] that solves approximately the optimal direct scheduling problem in general network topologies. Suppose that a set of packets and respective paths are given. The injection schedule is computed in polynomial time with respect to the size of the graph and the number of packets. The routing time is measured with respect to the *congestion* C of the packet paths (the maximum number of paths that use an edge), and the *dilation* D (the maximum length of any path).

The result in [6] establishes the existence of a simple greedy direct scheduling algorithm with routing time $rt = O(C \cdot D)$. In this algorithm, the packets are processed in an arbitrary order and each packet is assigned the smallest available injection time. The resulting routing time is worst-case optimal, since there exist instances of direct scheduling problems for which no direct scheduling algorithm can achieve a better routing time. A trivial lower bound on the routing time of any direct scheduling problem is $\Omega(C + D)$, since no algorithm can deliver the packets faster than the congestion or dilation of the paths. Thus, in the general case, the algorithm in [6] has routing time $rt = O((rt^*)^2)$, where rt^* is the optimal routing time.

Direct Routing in Specific Graphs

Several direct algorithms are presented in [6] for specialized network topologies. The algorithms solve the direct routing problem where first good paths are constructed and then an efficient injection schedule is computed. Given a set of packets, let C^* and D^* denote the optimal congestion and dilation, respectively, for all possible sets of paths for the packets. Clearly, the optimal routing time is $rt^* = \Omega(C^* + D^*)$. The upper bounds in the direct algorithm in [6] are expressed in terms of this lower bound. All the algorithms run in time polynomial to the size of the input.

Tree The graph G is an arbitrary tree. A direct routing algorithm is given in [Section 3.1 in 6], where each packet follows the shortest path from its source to the destination. The injection schedule is obtained using the greedy algorithm with a particular ordering of the packets. The routing time of the algorithm is asymptotically optimal: $rt \le 2C^* + D^* - 2 < 3 \cdot rt^*$.

Mesh The graph G is a d-dimensional mesh (grid) with n nodes [10]. A direct routing algorithm is proposed in [Section 3.2 in 6], which first constructs efficient paths for the packets with congestion $C = O(d \log n \cdot C^*)$ and dilation $D = O(d^2 \cdot D^*)$ (the congestion is guaranteed with high probability). Then, using these paths the injection schedule is computed giving a direct algorithm with the routing time:

$$rt = O(d^2 \log^2 n \cdot C^* + d^2 \cdot D^*) = O(d^2 \log^2 n \cdot rt^*) .$$

This result follows from a more general result which is shown in [6], that says that if the paths contain at most b "bends", i. e. at most b dimension changes, then there is a direct scheduling algorithm with routing time $O(b \cdot C + D)$. The result follows because the constructed paths have $b = O(d \log n)$ bends.

Butterfly The graph G is a butterfly network with n input and n output nodes [10]. In [Section 3.3 in 6] the authors examine permutation routing problems in the butterfly, where each input (output) node is the source (destination) of exactly one packet. An efficient direct routing algorithm is presented in [6] which first computes good paths for the packets using Valiant's method [14,15]: two butterflies are connected back to back, and each path is formed by choosing a random intermediate node in the output of the first butterfly. The chosen paths have congestion $C = O(\lg n)$ (with high probability) and dilation $D = 2 \lg n = O(D^*)$. Given the paths, there is a direct schedule with routing time very close to optimal: $rt \leq 5 \lg n = O(rt^*)$.

Hypercube The graph G is a hypercube with n nodes [10]. A direct routing algorithm is given in [Section 3.4 in 6] for permutation routing problems. The algorithm first computes good paths for the packets by selecting a single random intermediate node for each packet. Then an appropriate injection schedule gives routing time $rt < 14 \lg n$, which is worst-case optimal since there exist permutations for which $D^* = \Omega(\lg n)$.

Lower Bound for Buffering

In [Section 5 in 6] an additional problem has been studied about the amount of buffering required to provide small routing times. It is shown in [6] that there is a direct scheduling problem for which every direct algorithm requires routing time $\Omega(C \cdot D)$; at the same time, $C + D = \Theta(\sqrt{C \cdot D}) = o(C \cdot D)$. If buffering of packets is allowed, then it is well known that there exist packet scheduling algorithms ([11,12]) with routing time very close to the optimal $O(C + D)$. In [6] it is shown that for the particular packet problem, in order to convert a direct injection schedule of routing time $O(C \cdot D)$ to a packet schedule with routing time $O(C + D)$, it is necessary to buffer packets in the network nodes in total $\Omega(N^{4/3})$ times, where a packet buffering corresponds to keeping a packet in an intermediate node buffer for a time step, and N is the number of packets.

Related Work

The only previous work which specifically addresses direct routing is for permutation problems on trees [3,13]. In these papers, the resulting routing time is $O(n)$ for any tree with n nodes. This is worst-case optimal, while the result in [6] is asymptotically optimal for all routing problems in trees.

Cypher *et al.* [7] study an online version of direct routing in which a worm (packet of length L) can be retransmitted if it is dropped (they also allow the links to have bandwidth $B \geq 1$). Adler et al. [1] study time constrained direct routing, where the task is to schedule as many packets as possible within a given time frame.They show that the time constrained version of the problem is NP-complete, and also study approximation algorithms on trees and meshes. Further, they discuss how much buffering could help in this setting.

Other models of bufferless routing are *matching routing* [2] where packets move to their destinations by swapping packets in adjacent nodes, and *hot-potato routing* [4,5,8,9] in which packets follow links that bring them closer to the destination, and if they cannot move closer (due to collisions) they are deflected toward alternative directions.

Applications

Direct routing represent collision-free communication protocols, in which packets spend the smallest amount of time possible time in the network once they are injected. This type of routing is appealing in power or resource constrained environments, such as optical networks, where packet buffering is expensive, or sensor networks where energy resources are limited. Direct routing is also important for providing quality of service in networks. There exist applications where it is desirable to provide guarantees on the delivery time of the packets after they are injected into the network, for example in streaming audio and video. Direct routing is suitable for such applications.

Cross References

▶ Oblivious Routing
▶ Packet Routing

Recommended Reading

1. Adler, M., Khanna, S., Rajaraman, R., Rosén, A.: Time-constrained scheduling of weighted packets on trees and meshes. Algorithmica **36**, 123–152 (2003)
2. Alon, N., Chung, F., Graham, R.: Routing permutations on graphs via matching. SIAM J. Discret. Math. **7**(3), 513–530 (1994)
3. Alstrup, S., Holm, J., de Lichtenberg, K., Thorup, M.: Direct routing on trees. In: Proceedings of the Ninth Annual ACM-SIAM, Symposium on Discrete Algorithms (SODA 98), pp. 342–349. San Francisco, California, United States (1998)
4. Ben-Dor, A., Halevi, S., Schuster, A.: Potential function analysis of greedy hot-potato routing. Theor. Comput. Syst. **31**(1), 41–61 (1998)

5. Busch, C., Herlihy, M., Wattenhofer, R.: Hard-potato routing. In: Proceedings of the 32nd Annual ACM Symposium on Theory of Computing, pp. 278–285. Portland, Oregon, United States (2000)
6. Busch, C., Magdon-Ismail, M., Mavronicolas, M., Spirakis, P.: Direct routing: Algorithms and Complexity. Algorithmica **45**(1), 45–68 (2006)
7. Cypher, R., Meyer auf der Heide, F., Scheideler, C., Vöcking, B.: Universal algorithms for store-and-forward and wormhole routing. In: Proceedings of the 28th ACM Symposium on Theory of Computing, pp. 356–365. Philadelphia, Pennsylvania, USA (1996)
8. Feige, U., Raghavan, P.: Exact analysis of hot-potato routing. In: IEEE (ed.) Proceedings of the 33rd Annual, Symposium on Foundations of Computer Science, pp. 553–562, Pittsburgh (1992)
9. Kaklamanis, C., Krizanc, D., Rao, S.: Hot-potato routing on processor arrays. In: Proceedings of the 5th Annual ACM, Symposium on Parallel Algorithms and Architectures, pp. 273–282, Velen (1993)
10. Leighton, F.T.: Introduction to Parallel Algorithms and Architectures: Arrays – Trees – Hypercubes. Morgan Kaufmann, San Mateo (1992)
11. Leighton, F.T., Maggs, B.M., Rao, S.B.: Packet routing and job-scheduling in O(congestion+dilation) steps. Combinatorica **14**, 167–186 (1994)
12. Leighton, T., Maggs, B., Richa, A.W.: Fast algorithms for finding O(congestion + dilation) packet routing schedules. Combinatorica **19**, 375–401 (1999)
13. Symvonis, A.: Routing on trees. Inf. Process. Lett. **57**(4), 215–223 (1996)
14. Valiant, L.G.: A scheme for fast parallel communication. SIAM J. Comput. **11**, 350–361 (1982)
15. Valiant, L.G., Brebner, G.J.: Universal schemes for parallel communication. In: Proceedings of the 13th Annual ACM, Symposium on Theory of Computing, pp. 263–277. Milwaukee, Wisconsin, United States (1981)

Distance-Based Phylogeny Reconstruction (Fast-Converging)

2003; King, Zhang, Zhou

MIKLÓS CSŰRÖS
Department of Computer Science, University of Montreal, Montreal, QC, Canada

Keywords and Synonyms

Learning an evolutionary tree

Problem Definition

Introduction

From a mathematical point of view, a phylogeny defines a probability space for random sequences observed at the leaves of a binary tree T. The tree T represents the unknown hierarchy of common ancestors to the sequences. It is assumed that (unobserved) ancestral sequences are associated with the inner nodes. The tree along with the associated sequences models the evolution of a molecular sequence, such as the protein sequence of a gene. In the conceptually simplest case, each tree node corresponds to a species, and the gene evolves within the organismal lineages by vertical descent.

Phylogeny reconstruction consists of finding T from observed sequences. The possibility of such reconstruction is implied by fundamental principles of molecular evolution, namely, that random mutations within individuals at the genetic level spreading to an entire mating population are not uncommon, since often they hardly influence evolutionary fitness [15]. Such mutations slowly accumulate, and, thus, differences between sequences indicate their evolutionary relatedness.

The reconstruction is theoretically feasible in several known situations. In some cases, distances can be computed between the sequences, and used in a distance-based algorithm. Such an algorithm is fast-converging if it almost surely recovers T, using sequences that are polynomially long in the size of T. Fast-converging algorithms exploit statistical concentration properties of distance estimation.

Formal Definitions

An evolutionary *topology* $U(X)$ is an unrooted binary tree in which leaves are bijectively mapped to a set of species X. A *rooted topology* T is obtained by rooting a topology U on one of the edges uv: a new node ρ is added (the *root*), the edge uv is replaced by two edges ρv and ρu, and the edges are directed outwards on paths from ρ to the leaves. The edges, vertices, and leaves of a rooted or unrooted topology T are denoted by $\mathcal{E}(T)$, $\mathcal{V}(T)$ and $\mathcal{L}(T)$, respectively.

The edges of an unrooted topology U may be equipped with a a positive *edge length* function $d\colon E(U) \mapsto (0, \infty)$. Edge lengths induce a *tree metric* $d\colon \mathcal{V}(U) \times \mathcal{V}(U) \mapsto [0, \infty)$ by the extension $d(u, v) = \sum_{e \in u \rightsquigarrow v} d(e)$, where $u \rightsquigarrow v$ denotes the unique path from u to v. The value $d(u, v)$ is called the *distance* between u and v. The pairwise distances between leaves form a *distance matrix*.

An *additive tree metric* is a function $\delta\colon X \times X \mapsto [0, \infty)$ that is equivalent to the distance matrix induced by some topology $U(X)$ and edge lengths. In certain random models, it is possible to define an additive tree metric that can be estimated from dissimilarities between sequences observed at the leaves.

In a *Markov model of character evolution* over a rooted topology T, each node u has an associated *state*, which is a random variable $\xi(u)$ taking values over a fixed alphabet $\mathcal{A} = \{1, 2, \ldots r\}$. The vector of leaf states constitutes the *character* $\xi = \big(\xi(u) \colon u \in \mathcal{L}(T)\big)$. The states form a first-order Markov chain along every path. The joint distribution of the node states is specified by the marginal distribution of the root state, and the conditional probabilities $\mathbb{P}\{\xi(v) = b | \xi(u) = a\} = p_e(a \to b)$ on each edge e, called *edge transition probabilities*.

A *sample* of length ℓ consists of independent and identically distributed characters $\Xi = \big(\xi_i \colon i = 1, \ldots \ell\big)$. The *random sequence* associated with the leaf u is the vector $\Xi(u) = \big(\xi_i(u) \colon i = 1, \ldots \ell\big)$.

A *phylogeny reconstruction algorithm* is a function $\mathcal{F}$ mapping samples to unrooted topologies. The *success probability* is the probability that $\mathcal{F}(\Xi)$ equals the true topology.

Popular Random Models

Neyman Model [14] The edge transition probabilities are

$$p_e(a \to b) = \begin{cases} 1 - \mu_e & \text{if } a = b\,; \\ \frac{\mu_e}{r-1} & \text{if } a \neq b \end{cases}$$

with some edge-specific mutation probability $0 < \mu_e < 1 - 1/r$. The root state is uniformly distributed. A distance is usually defined by

$$d(u, v) = -\frac{r-1}{r} \ln\Big(1 - \frac{r}{r-1} \mathbb{P}\{\xi(u) \neq \xi(v)\}\Big).$$

General Markov Model There are no restrictions on the edge transition probabilities in the general Markov model. For identifiability [1,16], however, it is usually assumed that $0 < \det \mathbf{P}_e < 1$, where $\mathbf{P}_e$ is the stochastic matrix of edge transition probabilities. Possible distances in this model include the *paralinear* distance [12,1] and the *LogDet* distance [13,16]. This latter is defined by $d(u, v) = -\ln \det \mathbf{J}_{uv}$, where J_{uv} is the matrix of joint probabilities for $\xi(u)$ and $\xi(v)$.

It is often assumed in practice that sequence evolution is effected by a continuous-time Markov process operating on the edges. Accordingly, the edge length directly measures time. In particular, $\mathbf{P}_e = e^{\mathbf{Q}\cdot d(e)}$ on every edge e, where $\mathbf{Q}$ is the instantaneous rate matrix of the underlying process.

Key Results

It turns out that the hardness of reconstructing an unrooted topology U from distances is determined by its *edge depth* $\rho(U)$. Edge depth is defined as the smallest integer k for which the following holds. From each endpoint of every edge $e \in \mathcal{E}(U)$, there is a path leading to a leaf, which does not include e and has at most k edges.

Theorem 1 (Erdős, Steel, Székely, Warnow [6]) *If U has n leaves, then $\rho(U) \leq 1 + \log_2(n-1)$. Moreover, for almost all random n-leaf topologies under the uniform or Yule-Harding distributions, $\rho(U) \in O(\log\log n)$*

Theorem 2 (Erdős, Steel, Székely, Warnow [6]) *For the Neyman model, there exists a polynomial-time algorithm that has a success probability $(1-\delta)$ for random samples of length*

$$\ell = O\Big(\frac{\log n + \log \frac{1}{\delta}}{f^2(1-2g)^{4\rho+6}}\Big), \tag{1}$$

where $0 < f = \min_e \mu_e$ and $g = \max_e \mu_e < 1/2$ are extremal edge mutation probabilities, and ρ is the edge depth of the true topology.

Theorem 2 can be extended to the general Markov model with analogous success rates for LogDet distances [7], as well as to a number of other Markov models [2].

Equation (1) shows that phylogenies can be reconstructed with high probability from polynomially long sequences. Algorithms with such sample size requirements were dubbed *fast-converging* [9]. Fast convergence was proven for the short quartet methods of Erdős et al. [6,7], and for certain variants [11] of the so-called disk-covering methods introduced by Huson et al. [9]. All these algorithms run in $\Omega(n^5)$ time. Csürös and Kao [3] initiated the study of computationally efficient fast-converging algorithms, with a cubic-time solution. Csürös [2] gave a quadratic-time algorithm. King et al. [10] designed an algorithm with an optimal running time of $O(n \log n)$ for producing a phylogeny from a matrix of estimated distances.

The short quartet methods were revisited recently: [4] described an $O(n^4)$-time method that aims at succeeding even if only a short sample is available. In such a case, the algorithm constructs a forest of "trustworthy" edges that match the true topology with high probability.

All known fast-converging distance-based algorithms have essentially the same sample bound as in (1), but Daskalakis et al. [5] recently gave a twist to the notion of fast convergence. They described a polynomial-time algorithm, which outputs the true topology almost surely from a sample of size $O(\log n)$, given that edge lengths are not too large. Such a bound is asymptotically optimal [6]. Interestingly, the sample size bound does not involve exponential dependence on the edge depth: the algorithm does not rely on a distance matrix.

Applications

Phylogenies are often constructed in molecular evolution studies, from aligned DNA or protein sequences. Fast-converging algorithms have mostly a theoretical appeal at this point. Fast convergence promises a way to handle the increasingly important issue of constructing large-scale phylogenies: see, for example, the CIPRES project (http://www.phylo.org/).

Cross References

Similar algorithmic problems are discussed under the heading ► Distance-based phylogeny reconstruction (optimal radius).

Recommended Reading

Joseph Felsenstein wrote a definitive guide [8] to the methodology of phylogenetic reconstruction.

1. Chang, J.T.: Full reconstruction of Markov models on evolutionary trees: identifiability and consistency. Math. Biosci. **137**, 51–73 (1996)
2. Csürös, M.: Fast recovery of evolutionary trees with thousands of nodes. J. Comput. Biol. **9**(2), 277–297 (2002) Conference version at RECOMB 2001
3. Csürös, M., Kao, M.-Y.: Provably fast and accurate recovery of evolutionary trees through Harmonic Greedy Triplets. SIAM J. Comput. **31**(1), 306–322 (2001) Conference version at SODA (1999)
4. Daskalakis, C., Hill, C., Jaffe, A., Mihaescu, R., Mossel, E., Rao, S.: Maximal accurate forests from distance matrices. In: Proc. Research in Computational Biology (RECOMB), pp. 281–295 (2006)
5. Daskalakis, C., Mossel, E., Roch, S.: Optimal phylogenetic reconstruction. In: Proc. ACM Symposium on Theory of Computing (STOC), pp. 159–168 (2006)
6. Erdős, P.L., Steel, M.A., Székely, L.A., Warnow, T.J.: A few logs suffice to build (almost) all trees (I). Random Struct. Algorithm **14**, 153–184 (1999) Preliminary version as DIMACS TR97-71
7. Erdős, P.L., Steel, M.A., Székely, L. A., Warnow, T.J.: A few logs suffice to build (almost) all trees (II). Theor. Comput. Sci. **221**, 77–118 (1999) Preliminary version as DIMACS TR97-72
8. Felsenstein, J.: Inferring Pylogenies. Sinauer Associates, Sunderland, Massachusetts (2004)
9. Huson, D., Nettles, S., Warnow, T.: Disk-covering, a fast converging method of phylogenetic reconstruction. J. Comput. Biol. **6**(3–4) 369–386 (1999) Conference version at RECOMB (1999)
10. King, V., Zhang, L., Zhou, Y.: On the complexity of distance-based evolutionary tree reconstruction. In: Proc. ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 444–453 (2003)
11. Lagergren, J.: Combining polynomial running time and fast convergence for the disk-covering method. J. Comput. Syst. Sci. **65**(3), 481–493 (2002)
12. Lake, J.A.: Reconstructing evolutionary trees from DNA and protein sequences: paralinear distances. Proc. Natl. Acad. Sci. USA **91**, 1455–1459 (1994)
13. Lockhart, P.J., Steel, M.A., Hendy, M.D., Penny, D.: Recovering evolutionary trees under a more realistic model of sequence evolution. Mol. Biol. Evol. **11**, 605–612 (1994)
14. Neyman, J.: Molecular studies of evolution: a source of novel statistical problems. In: Gupta, S.S., Yackel, J. (eds) Statistical Decision Theory and Related Topics, pp. 1–27. Academic Press, New York (1971)
15. Ohta, T.: Near-neutrality in evolution of genes and gene regulation. Proc. Natl. Acad. Sci. USA **99**, 16134–16137 (2002)
16. Steel, M.A.: Recovering a tree from the leaf colourations it generates under a Markov model. Appl. Math. Lett. **7**, 19–24 (1994)

Distance-Based Phylogeny Reconstruction (Optimal Radius)

1999; Atteson
2005; Elias, Lagergren

RICHARD DESPER[1], OLIVIER GASCUEL[2]
[1] Department of Biology, University College London, London, UK
[2] LIRMM, National Scientific Research Center, Montpellier, France

Keywords and Synonyms

Phylogeny reconstruction; Distance methods; Performance analysis; Robustness; Safety radius approach; Optimal radius

Problem Definition

A phylogeny is an evolutionary tree tracing the shared history, including common ancestors, of a set of extant taxa. Phylogenies have been historically reconstructed using character-based (parsimony) methods, but in recent years the advent of DNA sequencing, along with the development of large databases of molecular data, has led to more involved methods. Sophisticated techniques such as likelihood and Bayesian methods are used to estimate phylogenies with sound statistical justifications. However, these statistical techniques suffer from the discrete nature of tree topology space. Since the number of tree topologies increases exponentially as a function of the number of taxa, and each topology requires separate likelihood calculation, it is important to restrict the search space and to design efficient heuristics. Distance methods for phylogeny reconstruction serve this purpose by inferring trees in a fraction of the time required for the more statistically rigorous methods. They allow dealing with thousands of taxa, while the current implementations of statistical approaches are limited to a few hundreds, and distance methods also provide fairly accurate starting trees to be further refined by more sophisticated methods. Moreover, the input of distance methods is the matrix of pairwise evolutionary distances among taxa, which are estimated by maximum likelihood, so that distance methods also have sound statistical justifications.

Mathematically, a phylogenetic tree is a triple $T = (V, E, l)$ where V is the set of nodes representing extant taxa and ancestral species, E is the set of edges (branches), and l is a function that assigns positive lengths to each edge in E. Evolution proceeds through the tree structure as a stochastic process with a finite state space corresponding to the DNA bases or amino acids present in the DNA or protein sequences, respectively.

Any phylogenetic tree T defines a metric D_T on its leaf set $L(T)$: let $P_T(u, v)$ define the unique path through T from u to v, then the distance from u to v is set to $D_T(u, v) = \sum_{e \in P_T(u,v)} l(e)$.

Distance methods for phylogeny reconstruction rely on the observation [13] that the map $T \to D_T$ is reversible; i. e., a tree T can be reconstructed from its tree metric. While in practice D_T is not known, by using models of evolution (e. g. [10], reviewed in [5]) one can use molecular sequence data to estimate a distance matrix D that approximates D_T. As the amount of sequence data increases, the consistency of the various models of sequence evolution implies that D should converge to D_T. Thus for a distance method to be *consistent*, it is necessary that for any tree T, and for distance matrices D “close enough” to D_T, the algorithm will output T.

The present chapter deals with the question of when any distance algorithm for phylogeny reconstruction can be guaranteed to output the correct phylogeny as a function of the divergence between the metric underlying the true phylogeny and the metric estimated from the data. Atteson [1] demonstrated that this consistency can be shown for Neighbor Joining (NJ) [11], the most popular distance method, and a number of NJ’s variants.

The Neighbor Joining (NJ) Algorithm of Saitou and Nei (1987)

NJ is *agglomerative*: it works by using the input matrix D to identify a pair of taxa $x, y \in L$ that are neighbors in T, i. e. there exists a node $u \in V$ such that $\{(u, x), (u, y)\} \subset E$. The algorithm creates a node c that is connected to x and y, extends the distance matrix to c, and then solves the reduced problem on $L \cup \{c\} \backslash \{x, y\}$. The pair (x, y) is chosen to minimize the following sum:

$$S_D(x, y) = (|L| - 2) \cdot D(x, y) - \sum_{z \in L} \big(D(z, x) + D(z, y)\big) .$$

The soundness of NJ is based on the observation that, if $D = D_T$ for a tree T, the value $S_D(x, y)$ will be minimized for a pair (x, y) that are neighbors in T. A number of papers (reviewed in [8]) have been dedicated to the various interpretations and properties of the S_D criterion.

The Fast Neighbor Joining (FNJ) Algorithm of Elias and Lagergren (2005)

NJ requires $\Omega(n^3)$ computations, where n is the number of taxa in the data set. Since a distance matrix only has n^2 entries, many attempts have been made to construct a distance algorithm that would only require $O(n^2)$ computations while retaining the accuracy of NJ. To this end, the best result so far is the Fast Neighbor Joining (FNJ) algorithm of Elias and Lagergren [4].

Most of the computation of NJ is used in the recalculations of the sums $S_D(x, y)$ after each agglomeration step. Although each recalculation can be performed in constant time, the number of such pairs is $\Omega(k^2)$ when k nodes are left to agglomerate, and thus, summing over k, $\Omega(n^3)$ computations are required in all.

Elias and Lagergren take a related approach to agglomeration, which does not exhaustively seek the minimum value of $S_D(x, y)$ at each step, but instead uses a heuristic to maintain a list of candidates of “visible pairs” (x, y) for agglomeration. At the $(n - k)^{th}$ step, when two neighbors are agglomerated from a k-taxa tree to form a (k-1)-taxa tree, FNJ has a list of $O(k)$ visible pairs for which $S_D(x, y)$ is calculated. The pair joined is selected from this list. By trimming the number of pairs considered, Elias and Lagergren achieved an algorithm which requires only $O(n^2)$ computations.

Safety Radius-Based Performance Analysis (Atteson 1999)

Short branches in a phylogeny are difficult to resolve, especially when they are nested deep within a tree, because relatively few mutations occurring on a short branch as opposed to on much longer pendant branches, which hides phylogenetic signal. One is faced with the choice between leaving certain evolutionary relationships unresolved (i. e., having an internal node with degree > 3), or examining when confidence can be had in the resolution of a short internal edge.

A natural formulation [9] of this question is: how long must be molecular sequences before one can have confidence in an algorithm’s ability to reconstruct T accurately? An alternative formulation [1] appropriate for distance methods: if D is a distance matrix that approximates a tree metric D_T, can one have some confidence in an algorithm’s ability to reconstruct T given D, based on some measure of the distance between D and D_T? For two matrices, D_1 and D_2, the L_∞ distance between them is defined by $\|D_1 - D_2\|_\infty = \max_{i,j} |D_1(i,j) - D_2(i,j)|$. Moreover, let $\mu(T)$ denote the length of the shortest internal edge of a tree T.

The latter formulation leads to a definition: The *safety radius* of an algorithm $\mathcal{A}$ is the greatest value of r with the property that: given any phylogeny T, and any distance matrix D satisfying $\|D - D_T\|_\infty < r \cdot \mu(T)$, $\mathcal{A}$ will return the tree T.

Key Results

Atteson [1] answered the second question affirmatively, with two theorems.

Theorem 1 *The safety radius of NJ is* 1/2.

Theorem 2 *For no distance algorithm $\mathcal{A}$ is the safety radius of $\mathcal{A}$ greater than* 1/2.

Indeed, given any μ, one can find two different trees T_1, T_2 and a distance matrix D such that $\mu = \mu(T_1) = \mu(T_2)$ and $\|D - D_{T_1}\|_\infty = \mu/2 = \|D - D_{T_2}\|_\infty$. Since D is equidistant from two distinct tree metrics, no algorithm could assign it to the "closest" tree.

In their presentation of an optimally fast version of the NJ algorithm, Elias and Lagergren updated Atteson's results for the FNJ algorithm. They showed

Theorem 3 *The safety radius of FNJ is* 1/2.

Elias and Lagergren showed that if D is a distance matrix and D_T is a tree metric with $\|D - D_T\|_\infty < \mu(T)/2$, then FNJ will output the same tree (T) as NJ.

Applications

Phylogeny is a quite active field within evolutionary biology and bioinformatics. As more proteins and DNA sequences become available, the need for fast and accurate phylogeny estimation algorithms is ever increasing as phylogeny not only serves to reconstruct species history but also to decipher genomes. To date, NJ remains one of the most popular algorithms for phylogeny building, and is by far the most popular of the distance methods, with well over 1000 citations per year.

Open Problems

With increasing amounts of sequence data becoming available for an increasing number of species, distance algorithms such as NJ should be useful for quite some time. Currently, the bottleneck in the process of building phylogenies is not the problem of searching topology space, but rather the problem of building distance matrices. The brute force method to build a distance matrix on n taxa from sequences with l positions requires $\Omega(ln^2)$ computations, and typically $l \gg n$. Elias and Lagergren proposed an $\Omega(ln^{1.376})$ algorithm based on Hamming distance and matrix calculations. However, this algorithm only applies to over-simple distance estimators [10]. Extending this result to more realistic models would be a great advance.

A number of distance-based tree building algorithms have been analyzed in the safety radius framework. Atteson [1] dealt with a large class of neighbor joining-like algorithms, and Gascuel and McKenzie [7] studied the ultrametric setting where the correct tree T is rooted and all tree leaves are at the same distance from the root. Such trees are very common; they are called "molecular clock" trees in phylogenetics and "indexed hierarchies" in data analysis. In this setting, the optimal safety radius is equal to 1 (instead of 1/2) and a number of standard algorithms (e. g. UPGMA, with time complexity in $O(n^2)$) have a safety radius of 1. However, experimental studies (see below) showed that not all algorithms with optimal safety radius achieve the same accuracy, indicating that the safety radius approach should be sharpened to provide better theoretical analysis of method performance.

Experimental Results

Computer simulation is the most standard way to assess algorithm accuracy in phylogenetics. A tree is randomly generated as well as a sequence at tree root, whose evolution is simulated along the tree edges. A reconstruction algorithm is tested using the sequences observed at the tree leaves, thus mimicking the phylogenetic task. Various measures exist to compare the correct and the inferred trees, and algorithm performance is assessed as the average measure over repeated experiments. Elias and Lagergren [4] showed that FNJ (in $O(n^2)$) is just slightly outperformed by NJ (in $O(n^3)$), while numerous simulations (e. g. [3,12]) indicated that NJ is beaten by more recent algorithms (all in $O(n^3)$ or less), namely BioNJ [6], WEIGHBOR [2], FastME [3] and STC [12].

Data Sets

A large number of data sets is stored by the TreeBASE project, at http://www.treebase.org.

URL to Code

For a list of leading phylogeny packages, see Joseph Felsenstein's website at http://evolution.genetics.washington.edu/phylip/software.html

Cross References

- ▶ Approximating Metric Spaces by Tree Metrics
- ▶ Directed Perfect Phylogeny (Binary Characters)
- ▶ Distance-Based Phylogeny Reconstruction (Fast-Converging)
- ▶ Perfect Phylogeny (Bounded Number of States)
- ▶ Perfect Phylogeny Haplotyping
- ▶ Phylogenetic Tree Construction from a Distance Matrix

Recommended Reading

1. Atteson, K.: The performance of neighbor-joining methods of phylogenetic reconstruction. Algorithmica **25**, 251–278 (1999)
2. Bruno, W.J., Socci, N.D., Halpern, A.L.: Weighted Neighbor Joining: A Likelihood-Based Approach to Distance-Based Phylogeny Reconstruction. Mol. Biol. Evol. **17**, 189–197 (2000)
3. Desper, R., Gascuel, O.: Fast and Accurate Phylogeny Reconstruction Algorithms Based on the Minimum – Evolution Principle. J. Comput. Biol. **9**, 687–706 (2002)
4. Elias, I. Lagergren, J.: Fast Neighbor Joining. In: Proceedings of the 32nd International Colloquium on Automata, Languages, and Programming (ICALP), pp. 1263–1274 (2005)
5. Felsenstein, J.: Inferring Phylogenies. Sinauer Associates, Sunderland, Massachusetts (2004)
6. Gascuel, O.: BIONJ: an Improved Version of the NJ Algorithm Based on a Simple Model of Sequence Data. Mol. Biol. Evol. **14**, 685–695 (1997)
7. Gascuel, O. McKenzie, A.: Performance Analysis of Hierarchical Clustering Algorithms. J. Classif. **21**, 3–18 (2004)
8. Gascuel, O., Steel, M.: Neighbor-Joining Revealed. Mol. Biol. Evol. **23**, 1997–2000 (2006)
9. Huson, D.H., Nettles, S., Warnow, T.: Disk-covering, a fast-converging method for phylogenetic tree reconstruction. J. Comput. Biol. **6**, 369–386 (1999)
10. Jukes, T.H., Cantor, C.R.: Evolution of Protein Molecules. In: Munro, H.N. (ed.), Mammalian Protein Metabolism, pp. 21–132, Academic Press, New York (1969)
11. Saitou, N., Nei, M.: The Neighbor-joining Method: A New Method for Reconstructing Phylogenetic Trees. Mol. Biol. Evol. **4**, 406–425 (1987)
12. Vinh, L.S., von Haeseler, A.: Shortest triplet clustering: reconstructing large phylogenies using representative sets. BMC Bioinformatics **6**, 92 (2005)
13. Zarestkii, K.: Reconstructing a tree from the distances between its leaves. Uspehi Mathematicheskikh Nauk **20**, 90–92 (1965) (in russian)

Distributed Algorithms for Minimum Spanning Trees

1983; Gallager, Humblet, Spira

SERGIO RAJSBAUM
Math Institute, National Autonomous University of Mexico, Mexico City, Mexico

Keywords and Synonyms

Minimum weight spanning tree

Problem Definition

Consider a communication network, modeled by an undirected weighted graph $G = (V, E)$, where $|V| = n$, $|E| = m$. Each vertex of V represents a processor of unlimited computational power; the processors have unique identity numbers (ids), and they communicate via the edges of E by sending messages to each other. Also, each edge $e \in E$ has associated a weight $w(e)$, known to the processors at the endpoints of e. Thus, a processor knows which edges are incident to it, and their weights, but it does not know any other information about G. The network is *asynchronous:* each processor runs at an arbitrary speed, which is independent of the speed of other processors. A processor may wake up spontaneously, or when it receives a message from another processor. There are no failures in the network. Each message sent arrives at its destination within a finite but arbitrary delay. A *distributed algorithm* A for G is a set of local algorithms, one for each processor of G, that include instructions for sending and receiving messages along the edges of the network. Assuming that A terminates (i. e. all the local algorithms eventually terminate), its *message complexity* is the total number of messages sent over any execution of the algorithm, in the worst case. Its *time complexity* is the worst case execution time, assuming processor steps take negligible time, and message delays are normalized to be at most 1 unit.

A *minimum spanning tree* (MST) of G is a subset E' of E such that the graph $T = (V, E')$ is a tree (connected and acyclic) and its total weight, $w(E') = \sum_{e \in E'} w(e)$ is as small as possible. The computation of an MST is a central problem in combinatorial optimization, with a rich history dating back to 1926 [2], and up to now; the book [12] collects properties, classical results, applications, and recent research developments.

In the *distributed MST problem* the goal is to design a distributed algorithm A that terminates always, and computes an MST T of G. At the end of an execution, each processor knows which of its incident edges belong to the tree T and which not (i. e. the processor writes in a local output register the corresponding incident edges). It is remarkable that in the distributed version of the MST problem, a communication network is solving a problem where the input is the network itself. This is one of the fundamental starting points of network algorithms.

It is not hard to see that if all edge weights are different, the MST is unique. Due to the assumption that processors have unique ids, it is possible to assume that all edge weights are different: whenever two edge weights are equal, ties are broken using the processor ids of the edge endpoints. Having a unique MST facilitates the design of distributed algorithms, as processors can locally select edges that belong to the unique MST. Notice that if processors do not have unique ids, and edge weights are not different, there is no deterministic MST (nor any spanning tree) distributed algorithm, because it may be impossible to break the symmetry of the graph, for example, in the case it is a cycle with all edge weights equal.

Key Results

The distributed MST problem has been studied since 1977, and dozens of papers have been written on the subject. In 1983, the fundamental distributed *GHS algorithm* in [5] was published, the first to solve the MST problem with $O(m + n \log n)$ message complexity. The paper has had a very significant impact on research in distributed computing and won the 2004 Edsger W. Dijkstra Prize in Distributed Computing.

It is not hard to see that any distributed MST algorithm must have $\Omega(m)$ message complexity (intuitivelly, at least one message must traverse each edge). Also, results in [3,4] imply an $\Omega(n \log n)$ message complexity lower bound for the problem. Thus, the GHS algorithm is optimal in terms of message complexity.

The $\Omega(m + n \log n)$ message complexity lower bound for the construction of an MST applies also to the problem of finding an arbitrary spanning tree of the graph. However, for specific graph topologies, it may be easier to find an arbitrary spanning tree than to find an MST. In the case of a complete graph, $\Omega(n^2)$ messages are necessary to construct an MST [8], while an arbitrary spanning tree can be constructed in $O(n \log n)$ messages [7].

The time complexity of the GHS algorithm is $O(n \log n)$. In [1] it is described how to improve its time complexity to $O(n)$, while keeping the optimal $O(m + n \log n)$ message complexity. It is clear that $\Omega(D)$ time is necessary for the construction of a spanning tree, where D is the diameter of the graph. And in the case of an MST the time complexity may depend on other parameters of the graph. For example, due to the need for information flow among processors residing on a common cycle, as in an MST construction, at least one edge of the cycle must be excluded from the MST. If messages of unbounded size are allowed, an MST can be easily constructed in $O(D)$ time, by collecting the graph topology and edge weights in a root processor. The problem becomes interesting in the more realistic model where messages are of size $O(\log n)$, and an edge weight can be sent in a single message. When the number of messages is not important, one can assume without loss of generality that the model is synchronous. For near time optimal algorithms and lower bounds see [10] and references herein.

Applications

The distributed MST problem is important to solve, both theoretically and practically, as an MST can be used to save on communication, in various tasks such as broadcast and leader election, by sending the messages of such applications over the edges of the MST.

Also, research on the MST problem, and in particular the MST algorithm of [5], has motivated a lot of work. Most notably, the algorithm of [5], introduced various techniques that have been in widespread use for multicasting, query and reply, cluster coordination and routing, protocols for handshake, synchronization, and distributed phases. Although the algorithm is intuitive and is easy to comprehend, it is sufficiently complicated and interesting that it has become a challenge problem for formal verification methods e. g. [11].

Open Problems

There are many open problems in this area, only a few significant ones are mentioned. As far as message complexity, although the asymptotically tight bound of $O(m + n \log n)$ for the MST problem in general graphs is known, finding the actual constants remains an open problem. There are smaller constants known for general spanning trees than for MST though [6].

As mentioned above, near time optimal algorithms and lower bounds appear in [10] and references herein. The optimal time complexity remains an open problem. Also, in a synchronous model for overlay networks, where all processors are directly connected to each other, an MST can be constructed in sublogarithmic time, namely $O(\log \log n)$ communication rounds [9], and no corresponding lower bound is known.

Cross References

▶ Synchronizers, Spanners

Recommended Reading

1. Awerbuch, B.: Optimal distributed algorithms for minimum weight spanning tree, counting, leader election and related problems (detailed summary). In: Proc. of the 19th Annual ACM Symposium on Theory of Computing, pp. 230–240. ACM, USA (1987)

2. Borůvka, O.: Otakar Borůvka on minimum spanning tree problem (translation of both the 1926 papers, comments, history). Disc. Math. **233**, 3–36 (2001)
3. Burns, J.E.: A formal model for message-passing systems. Indiana University, Bloomington, TR-91, USA (1980)
4. Frederickson, G., Lynch, N.: The impact of synchronous communication on the problem of electing a leader in a ring. In: Proc. of the 16th Annual ACM Symposium on Theory of Computing, pp. 493–503. ACM, USA (1984)
5. Gallager, R.G., Humblet, P.A., Spira, P.M.: A distributed algorithm for minimum-weight spanning trees. ACM Trans. Prog. Lang. Systems **5**(1), 66–77 (1983)
6. Johansen, K.E., Jorgensen, U.L., Nielsen, S.H.: A distributed spanning tree algorithm. In: Proc. 2nd Int. Workshop on Distributed Algorithms (DISC). Lecture Notes in Computer Science, vol. 312, pp. 1–12. Springer, Berlin Heidelberg (1987)
7. Korach, E., Moran, S., Zaks, S.: Tight upper and lower bounds for some distributed algorithms for a complete network of processors. In: Proc. 3rd Symp. on Principles of Distributed Computing (PODC), pp. 199–207. ACM, USA (1984)
8. Korach, E., Moran, S., Zaks, S.: The optimality of distributive constructions of minimum weight and degree restricted spanning trees in a complete network of processors. In: Proc. 4th Symp. on Principles of Distributed Computing (PODC), pp. 277–286. ACM, USA (1985)
9. Lotker, Z., Patt-Shamir, B., Pavlov, E., Peleg, D.: Minimum-weight spanning tree construction in $O(\log\log n)$ communication rounds. SIAM J. Comput. **35**(1), 120–131 (2005)
10. Lotker, Z., Patt-Shamir, B., Peleg, D.: Distributed MST for constant diameter graphs. Distrib. Comput. **18**(6), 453–460 (2006)
11. Moses, Y., Shimony, B.: A new proof of the GHS minimum spanning tree algorithm. In: Distributed Computing, 20th Int. Symp. (DISC), Stockholm, Sweden, September 18–20, 2006. Lecture Notes in Computer Science, vol. 4167, pp. 120–135. Springer, Berlin Heidelberg (2006)
12. Wu, B.Y., Chao, K.M.: Spanning Trees and Optimization Problems (Discrete Mathematics and Its Applications). Chapman Hall, USA (2004)

Distributed Computing

- ▶ Distributed Vertex Coloring
- ▶ Failure Detectors
- ▶ Mobile Agents and Exploration
- ▶ Optimal Probabilistic Synchronous Byzantine Agreement
- ▶ P2P
- ▶ Set Agreement

Distributed Vertex Coloring

2004; Finocchi, Panconesi, Silvestri

DEVDATT DUBHASHI
Department of Computer Science, Chalmers University of Technology and Gothenburg University, Gothenburg, Sweden

Keywords and Synonyms

Vertex coloring; Distributed computation

Problem Definition

The *vertex coloring problem* takes as input an undirected graph $G := (V, E)$ and computes a *vertex coloring*, i. e. a function, $c : V \to [k]$ for some positive integer k such that adjacent vertices are assigned different colors (that is, $c(u) \neq c(v)$ for all $(u, v) \in E$). In the $(\Delta + 1)$ vertex coloring problem, k is set equal to $\Delta + 1$ where Δ is the maximum degree of the input graph G. In general, $(\Delta + 1)$ colors could be necessary as the example of a clique shows. However, if the graph satisfies certain properties, it may be possible to find colorings with far fewer colors. Finding the minimum number of colors possible is a computationally hard problem: the corresponding decision problems are NP-complete [5]. In Brooks–Vizing colorings, the goal is to try to find colorings that are near optimal.

In this paper, the model of computation used is the synchronous, message passing framework as used in standard distributed computing [11]. The goal is then to describe very simple algorithms that can be implemented easily in this distributed model that simultaneously are *efficient* as measured by the number of rounds required and have good performance *quality* as measured by the number of colors used. For efficiency, the number of rounds is require to be poly-logarithmic in n, the number of vertices in the graph and for performance quality, the number of colors used is should be near-optimal.

Key Results

Key theoretical results related to distributed $(\Delta + 1)$-vertex coloring are due to Luby [9] and Johansson [7]. Both show how to compute a $(\Delta + 1)$-coloring in $O(\log n)$ rounds with high probability. For Brooks–Vizing colorings, Kim [8] showed that if the graph is square or triangle free, then it is possible to color it with $O(\Delta/\log \Delta)$ colors. If, moreover, the graph is regular of sufficiently high degree ($\Delta \gg \lg n$), then Grable and Panconesi [6] show how to color it with $O(\Delta/\log \Delta)$ colors in $O(\log n)$ rounds. See [10] for a comprehensive discussion of probabilistic techniques to achieve such colorings.

The present paper makes a comprehensive experimental analysis of distributed vertex coloring algorithms of the kind analyzed in these papers on various classes of graphs. The results are reported in Sect. "Experimental Results" below and the data sets used are described in Sect. "Data Sets".

Applications

Vertex coloring is a basic primitive in many applications: classical applications are *scheduling problems* involving a number of pairwise restrictions on which jobs can be done simultaneously. For instance, in attempting to schedule classes at a university, two courses taught by the same faculty member cannot be scheduled for the same time slot. Similarly, two course that are required by the same group of students also should not conflict. The problem of determining the minimum number of time slots needed subject to these restrictions can be cast as a vertex coloring problem. One very active application for vertex coloring is *register allocation*. The register allocation problem is to assign variables to a limited number of hardware registers during program execution. Variables in registers can be accessed much quicker than those not in registers. Typically, however, there are far more variables than registers so it is necessary to assign multiple variables to registers. Variables conflict with each other if one is used both before and after the other within a short period of time (for instance, within a subroutine). The goal is to assign variables that do not conflict so as to minimize the use of non-register memory. A simple approach to this is to create a graph where the nodes represent variables and an edge represents conflict between its nodes. A coloring is then a conflict-free assignment. If the number of colors used is less than the number of registers then a conflict-free register assignment is possible. Modern applications include *assigning frequencies* to mobile radios and other users of the electro-magnetic spectrum. In the simplest case, two customers that are sufficiently close must be assigned different frequencies, while those that are distant can share frequencies. The problem of minimizing the number of frequencies is then a vertex coloring problem For more applications and references, see Michael Trick's coloring page [12].

Open Problems

The experimental analysis shows convincingly and rather surprisingly that the simplest, trivial, version of the algorithm actually performs best uniformly! In particular,it significantly outperforms the algorithms which have been analyzed rigorously. The authors give some heuristic recurrences that describe the performance of the trivial algorithm. It is a challenging and interesting open problem to give a rigorous justification of these recurrences. Alternatively, and less appealing, a rigorous argument that shows that the trivial algorithm dominates the ones analyzed by Luby and Johansson is called for. Other issues about how local structure of the graph impacts on the performance of such algorithms (which is hinted at in the paper) is worth subjecting to further experimental and theoretical analysis.

Experimental Results

All the algorithms analyzed start by assigning an initial *palette* of colors to each vertex, and then repeating the following simple iteration round:

1. *Wake up!*: Each vertex independently of the others wakes up with a certain probability to participate in the coloring in this round.
2. *Try!*: Each vertex independently of the others, selects a tentative color from its palette of colors at this round.
3. *Resolve conflicts!*: If no neighbor of a vertex selects the same tentative color, then this color becomes final. Such a vertex exits the algorithm, and the remaining vertices update their palettes accordingly. If there is a conflict, then it is resolved in one of two ways: Either all conflicting vertices are deemed unsuccessful and proceed to the next round, or an independent set is computed, using the so-called Hungarian heuristic, amongst all the vertices that chose the same color. The vertices in the independent set receive their final colors and exit. The Hungarian heuristic for independent set is to consider the vertices in random order, deleting all neighbors of an encountered vertex which itself is added to the independent set, see [1, p. 91] for a cute analysis of this heuristic to prove Turan's Theorem.
4. *Feed the Hungry!*: If a vertex runs out of colors in its palette, then fresh new colors are given to it.

Several parameters can be varied in this basic scheme: the wake up probability, the conflict resolution and the size of the initial palette are the most important ones.

In $(\Delta + 1)$-coloring, the initial palette for a vertex v is set to $[\Delta] := \{1, \cdots, \Delta + 1\}$ (global setting) or $[d(v) + 1]$ (where $d(v)$ is the degree of vertex v) (local setting). The experimental results indicate that (a) the best wake-up probability is 1, (b) the local palette version is as good as the global one in running time, but can achieve significant color savings and (c) the Hungarian heuristic can be used with vertex identities rather than random numbers giving good results.

In the Brooks–Vizing colorings, the initial palette is set to $[d(v)/s]$ where s is a *shrinking factor*. The experimental results indicate that uniformly, the best algorithm is the one where the wake-up probability is 1, and conflicts are resolved by the Hungarian heuristic. This is both with respect to the running time, as well as the number of colors used. Realistically useful values of s are between 4 and 6 resulting in Δ/s-colorings. The running time performance is excellent, with even graphs with a thousand vertices colored within 20–30 rounds. When compared to the best sequential algorithms, these algorithms use between twice or thrice as many colors, but are much faster.

Data Sets

Test data was both generated synthetically using various random graph models, and benchmark real life test sets from the second DIMACS implementation challenge [3] and Joe Culberson's web-site [2] were also used.

Cross References

► Graph Coloring
► Randomization in Distributed Computing
► Randomized Gossiping in Radio Networks

Recommended Reading

1. Alon, N., Spencer, J.: The Probabilistic Method. Wiley (2000)
2. Culberson, J.C.: http://web.cs.ualberta.ca/~joe/Coloring/index.html
3. Ftp site of DIMACS implementation challenges, ftp://dimacs.rutgers.edu/pub/challenge/
4. Finocchi, I., Panconesi, A., Silvestri, R.: An experimental Analysis of Simple Distributed Vertex Coloring Algorithms. Algorithmica **41**, 1–23 (2004)
5. Garey, M., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-completeness. W.H. Freeman (1979)
6. Grable, D.A., Panconesi, A.: Fast distributed algorithms for Brooks–Vizing colorings. J. Algorithms **37**, 85–120 (2000)
7. Johansson, Ö.: Simple distributed $(\Delta + 1)$-coloring of graphs. Inf. Process. Lett. **70**, 229–232 (1999)
8. Kim, J.-H.: On Brook's Theorem for sparse graphs. Combin. Probab. Comput. **4**, 97–132 (1995)
9. Luby, M.: Removing randomness in parallel without processor penalty. J. Comput. Syst. Sci. **47**(2), 250–286 (1993)
10. Molly, M., Reed, B.: Graph Coloring and the Probabilistic method. Springer (2002)
11. Peleg, D.: Distributed Computing: A Locality-Sensitive Approach. In: SIAM Monographs on Discrete Mathematics and Applications 5 (2000)
12. Trick, M.: Michael Trick's coloring page: http://mat.gsia.cmu.edu/COLOR/color.html

Dominating Set

► Data Reduction for Domination in Graphs
► Greedy Set-Cover Algorithms

Dynamic Problems

► Fully Dynamic Connectivity
► Robust Geometric Computation
► Voltage Scheduling

Dynamic Trees

2005; Tarjan, Werneck

RENATO F. WERNECK
Microsoft Research Silicon Valley, La Avenida, CA, USA

Keywords and Synonyms

Link-cut trees

Problem Definition

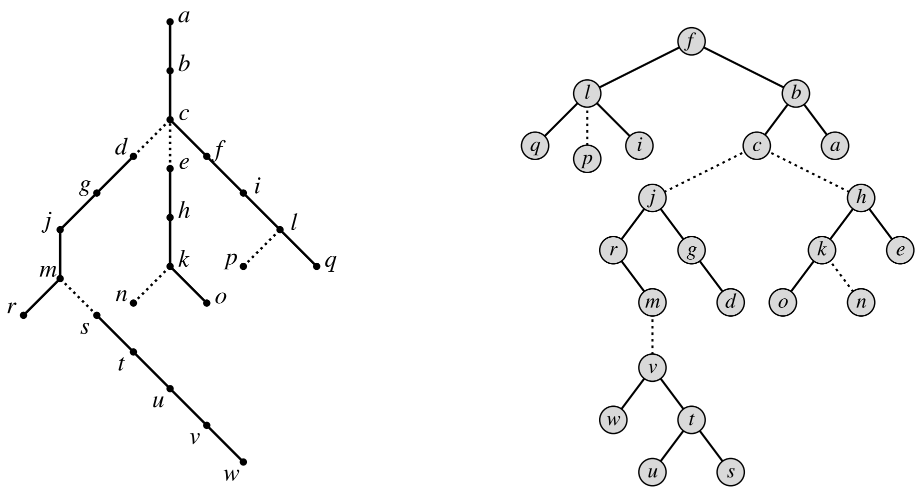
The *dynamic tree problem* is that of maintaining an arbitrary n-vertex forest that changes over time through edge insertions (*links*) and deletions (*cuts*). Depending on the application, one associates information with vertices, edges, or both. Queries and updates can deal with individual vertices or edges, but more commonly they refer to entire paths or trees. Typical operations include finding the minimum-cost edge along a path, determining the minimum-cost vertex in a tree, or adding a constant value to the cost of each edge on a path (or of each vertex of a tree). Each of these operations, as well as *links* and *cuts*, can be performed in $O(\log n)$ time with appropriate data structures.

Key Results

The obvious solution to the dynamic tree problem is to represent the forest explicitly. This, however, is inefficient for queries dealing with entire paths or trees, since it would require actually traversing them. Achieving $O(\log n)$ time per operation requires mapping each (possibly unbalanced) input tree into a balanced tree, which is better suited to maintaining information about paths or trees implicitly. There are three main approaches to perform the mapping: path decomposition, tree contraction, and linearization.

Path Decomposition

The first efficient dynamic tree data structure was Sleator and Tarjan's *ST-trees* [13,14], also known as *link-cut trees* or simply *dynamic trees*. They are meant to represent rooted trees, but the user can change the root with the *evert* operation. The data structure partitions each input tree into vertex-disjoint paths, and each path is represented as a binary search tree in which vertices appear in symmetric order. The binary trees are then connected according to how the paths are related in the forest. More precisely, the root of a binary tree becomes a *middle child* (in the data structure) of the parent (in the forest) of the topmost



Dynamic Trees, Figure 1
An ST-tree (adapted from [14]). On the *left*, the original tree, rooted at *a* and already partitioned into paths; on the *right*, the actual data structure. *Solid edges* connect nodes on the same path; *dashed edges* connect different paths

vertex of the corresponding path. Although a node has no more than two children (left and right) within its own binary tree, it may have arbitrarily many middle children. See Fig. 1. The path containing the root (*qlifcba* in the example) is said to be *exposed*, and is represented as the topmost binary tree. All path-related queries will refer to this path. The *expose* operation can be used to make any vertex part of the exposed path.

With standard balanced binary search trees (such as red-black trees), ST-trees support each dynamic tree operation in $O(\log^2 n)$ amortized time. This bound can be improved to $O(\log n)$ amortized with locally biased search trees, and to $O(\log n)$ in the worst case with globally biased search trees. Biased search trees (described in [5]), however, are notoriously complicated. A more practical implementation of ST-trees uses *splay trees*, a self-adjusting type of binary search trees, to support all dynamic tree operations in $O(\log n)$ amortized time [14].

Tree Contraction

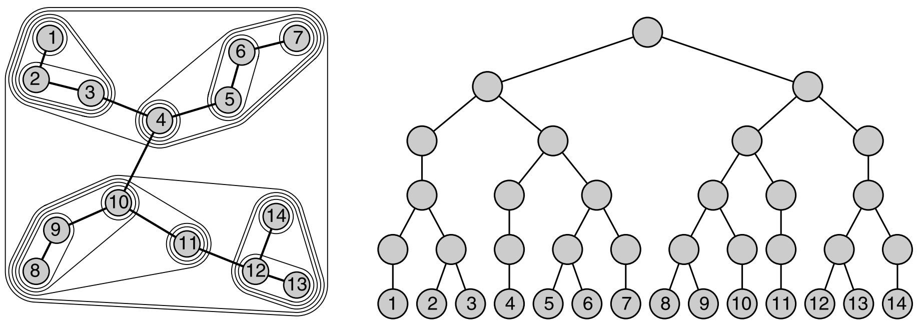
Unlike ST-trees, which represent the input trees directly, Frederickson's *topology trees* [6,7,8] represent a *contraction* of each tree. The original vertices constitute level 0 of the contraction. Level 1 represents a partition of these vertices into *clusters*: a degree-one vertex can be combined with its only neighbor; vertices of degree two that are adjacent to each other can be clustered together; other vertices are kept as singletons. The end result will be a smaller tree, whose own partition into clusters yields level 2. The process is repeated until a single cluster remains. The topology tree is a representation of the contraction, with each cluster having as children its constituent clusters on the level below. See Fig. 2.

With appropriate pieces of information stored in each cluster, the data structure can be used to answer queries about the entire tree or individual paths. After a *link* or *cut*, the affected topology trees can be rebuilt in $O(\log n)$ time.

The notion of tree contraction was developed independently by Miller and Reif [11] in the context of parallel algorithms. They propose two basic operations, *rake* (which eliminates vertices of degree one) and *compress* (which eliminates vertices of degree two). They show that $O(\log n)$ rounds of these operations are sufficient to contract any tree to a single cluster. Acar et al. translated a variant of their algorithm into a dynamic tree data structure, *RC-trees* [1], which can also be seen as a randomized (and simpler) version of topology trees.

A drawback of topology trees and RC-trees is that they require the underlying forest to have vertices with bounded (constant) degree in order to ensure $O(\log n)$ time per operation. Similarly, although ST-trees do not have this limitation when aggregating information over paths, they require bounded degrees to aggregate over trees. Degree restrictions can be addressed by "ternarizing" the input forest (replacing high-degree vertices with a series of low-degree ones [9]), but this introduces a host of special cases.

Alstrup et al.'s *top trees* [3,4] have no such limitation, which makes them more generic than all data structures previously discussed. Although also based on tree con-

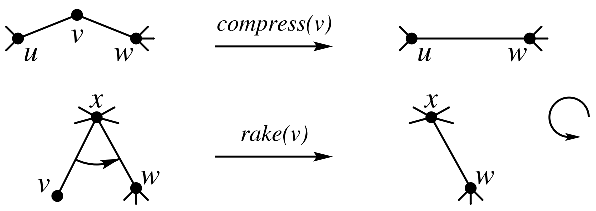


Dynamic Trees, Figure 2
A topology tree (adapted from [7]). On the *left*, the original tree and its multilevel partition; on the *right*, a corresponding topology tree

traction, their clusters behave not like vertices, but like *edges*. A *compress* cluster combines two edges that share a degree-two vertex, while a *rake* cluster combines an edge with a degree-one endpoint with a second edge adjacent to its other endpoint. See Fig. 3.

Top trees are designed so as to completely hide from the user the inner workings of the data structure. The user only specifies what pieces of information to store in each cluster, and (through call-back functions) how to update them after a cluster is created or destroyed when the tree changes. As long as the operations are properly defined, applications that use top trees are completely independent of how the data structure is actually implemented, i. e., of the order in which *rakes* and *compresses* are performed.

In fact, top trees were not even proposed as stand-alone data structures, but rather as an interface on top of topology trees. For efficiency reasons, however, one would rather have a more direct implementation. Holm, Tarjan, Thorup and Werneck have presented a conceptually simple stand-alone algorithm to update a top tree after a *link* or *cut* in $O(\log n)$ time in the worst case [17]. Tarjan and Werneck [16] have also introduced *self-adjusting top trees*, a more efficient implementation of top trees based on path decomposition: it partitions the input forest into edge-disjoint paths, represents these paths as splay trees, and connects these trees appropriately. Internally, the data structure is very similar to ST-trees, but the paths are edge-disjoint (instead of vertex-disjoint) and the ternarization step is incorporated into the data structure itself. All the user sees, however, are the *rakes* and *compresses* that characterize tree contraction.



Dynamic Trees, Figure 3
The *rake* and *compress* operations, as used by top trees (from [16])

Linearization

ET-trees, originally proposed by Henzinger and King [10] and later slightly simplified by Tarjan [15], use yet another approach to represent dynamic trees: *linearization*. It maintains an *Euler tour* of the each input tree, i. e., a closed path that traverses each edge twice—once in each direction. The tour induces a linear order among the vertices and arcs, and therefore can be represented as a balanced binary search tree. Linking and cutting edges from the forest corresponds to joining and splitting the affected binary trees, which can be done in $O(\log n)$ time. While linearization is arguably the simplest of the three approaches, it has a crucial drawback: because each edge appears twice, the data structure can only aggregate information over trees, not paths.

Lower Bounds

Dynamic tree data structures are capable of solving the *dynamic connectivity* problem on acyclic graphs: given two vertices v and w, decide whether they belong to the same tree or not. Pătraşcu and Demaine [12] have proven a lower bound of $\Omega(\log n)$ for this problem, which is matched by the data structures presented here.

Applications

Sleator and Tarjan's original application for dynamic trees was Dinic's blocking flow algorithm [13]. Dynamic trees are used to maintain a forest of arcs with positive residual capacity. As soon as the source *s* and the sink *t* become part of the same tree, the algorithm sends as much flow as possible along the *s*-*t* path; this reduces to zero the residual capacity of at least one arc, which is then *cut* from the tree. Several maximum flow and minimum-cost flow algorithms incorporating dynamic trees have been proposed ever since (some examples are [9,15]). Dynamic tree data structures, especially those based on tree contraction, are also commonly used within dynamic graph algorithms, such as the dynamic versions of minimum spanning trees [6,10], connectivity [10], biconnectivity [6], and bipartiteness [10]. Other applications include the evaluation of dynamic expression trees [8] and standard graph algorithms [13].

Experimental Results

Several studies have compared the performance of different dynamic-tree data structures; in most cases, ST-trees implemented with splay trees are the fastest alternative. Frederickson, for example, found that topology trees take almost 50% more time than splay-based ST-trees when executing dynamic tree operations within a maximum flow algorithm [8]. Acar et al. [2] have shown that RC-trees are significantly slower than splay-based ST-trees when most operations are *links* and *cuts* (such as in network flow algorithms), but faster when queries and value updates are dominant. The reason is that splaying changes the structure of ST-trees even during queries, while RC-trees remain unchanged.

Tarjan and Werneck [17] have presented an experimental comparison of several dynamic tree data structures. For random sequences of *links* and *cuts*, splay-based ST-trees are the fastest alternative, followed by splay-based ET-trees, self-adjusting top trees, worst-case top trees, and RC-trees. Similar relative performance was observed in more realistic sequences of operations, except when queries far outnumber structural operations; in this case, the self-adjusting data structures are slower than RC-trees and worst-case top trees. The same experimental study also considered the "obvious" implementation of ST-trees, which represents the forest explicitly and require linear time per operation in the worst case. Its simplicity makes it significantly faster than the $O(\log n)$-time data structures for path-related queries and updates, unless paths are hundred nodes long. The sophisticated solutions are more useful when the underlying forest has high diameter or there is a need to aggregate information over trees (and not only paths).

Cross References

► Fully Dynamic Connectivity
► Fully Dynamic Connectivity: Upper and Lower Bounds
► Fully Dynamic Higher Connectivity
► Fully Dynamic Higher Connectivity for Planar Graphs
► Fully Dynamic Minimum Spanning Trees
► Fully Dynamic Planarity Testing
► Lower Bounds for Dynamic Connectivity
► Routing

Recommended Reading

1. Acar, U.A., Blelloch, G.E., Harper, R., Vittes, J.L., Woo, S.L.M.: Dynamizing static algorithms, with applications to dynamic trees and history independence. In: Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 524–533. SIAM (2004)
2. Acar, U.A., Blelloch, G.E., Vittes, J.L.: An experimental analysis of change propagation in dynamic trees. In: Proceedings of the 7th Workshop on Algorithm Engineering and Experiments (ALENEX), pp. 41–54 (2005)
3. Alstrup, S., Holm, J., de Lichtenberg, K., Thorup, M.: Minimizing diameters of dynamic trees. In: Proceedings of the 24th International Colloquium on Automata, Languages and Programming (ICALP), Bologna, Italy, 7–11 July 1997. Lecture Notes in Computer Science, vol. 1256, pp. 270–280. Springer (1997)
4. Alstrup, S., Holm, J., Thorup, M., de Lichtenberg, K.: Maintaining information in fully dynamic trees with top trees. ACM Trans. Algorithms **1**(2), 243–264 (2005)
5. Bent, S.W., Sleator, D.D., Tarjan, R.E.: Biased search trees. SIAM J. Comput. **14**(3), 545–568 (1985)
6. Frederickson, G.N.: Data structures for on-line update of minimum spanning trees, with applications. SIAM J. Comput. **14**(4), 781–798 (1985)
7. Frederickson, G.N.: Ambivalent data structures for dynamic 2-edge-connectivity and *k* smallest spanning trees. SIAM J. Comput. **26**(2), 484–538 (1997)
8. Frederickson, G.N.: A data structure for dynamically maintaining rooted trees. J. Algorithms **24**(1), 37–65 (1997)
9. Goldberg, A.V., Grigoriadis, M.D., Tarjan, R.E.: Use of dynamic trees in a network simplex algorithm for the maximum flow problem. Math. Progr. **50**, 277–290 (1991)
10. Henzinger, M.R., King, V.: Randomized fully dynamic graph algorithms with polylogarihmic time per operation. In: Proceedings of the 27th Annual ACM Symposium on Theory of Computing (STOC), pp. 519–527 (1997)
11. Miller, G.L., Reif, J.H.: Parallel tree contraction and its applications. In: Proceedings of the 26th Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp. 478–489 (1985)
12. Pătraşcu, M., Demaine, E.D.: Lower bounds for dynamic connectivity. In: Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC), pp. 546–553 (2004)
13. Sleator, D.D., Tarjan, R.E.: A data structure for dynamic trees. J. Comput. Syst. Sci. **26**(3), 362–391 (1983)
14. Sleator, D.D., Tarjan, R.E.: Self-adjusting binary search trees. J. ACM **32**(3), 652–686 (1985)

15. Tarjan, R.E.: Dynamic trees as search trees via Euler tours, applied to the network simplex algorithm. Math. Prog. **78**, 169–177 (1997)
16. Tarjan, R.E., Werneck, R.F.: Self-adjusting top trees. In: Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 813–822 (2005)
17. Tarjan, R.E., Werneck, R.F.: Dynamic trees in practice. In: Proceedings of the 6th Workshop on Experimental Algorithms (WEA). Lecture Notes in Computer Science, vol. 4525, pp. 80–93 (2007)
18. Werneck, R.F.: Design and Analysis of Data Structures for Dynamic Trees. Ph. D. thesis, Princeton University (2006)

E

Edit Distance Under Block Operations

2000; Cormode, Paterson, Sahinalp, Vishkin
2000; Muthukrishnan, Sahinalp

S. CENK SAHINALP
Lab for Computational Biology, Simon Fraser University, Burnaby, BC, USA

Keywords and Synonyms

Block edit distance

Problem Definition

Given two strings $S = s_1 s_2 \dots s_n$ and $R = r_1 r_2 \dots r_m$ (wlog let $n \geq m$) over an alphabet $\sigma = \{\sigma_1, \sigma_2, \dots \sigma_\ell\}$, the *standard edit distance* between S and R, denoted $ED(S, R)$ is the minimum number of *single character edits*, specifically *insertions*, *deletions* and *replacements*, to transform S into R (equivalently R into S).

If the input strings S and R are permutations of the alphabet σ (so that $|S| = |R| = |\sigma|$) then an analogous *permutation edit distance* between S and R, denoted $PED(S, R)$ can be defined as the minimum number of single character *moves*, to transform S into R (or vice versa).

A generalization of the standard edit distance is *edit distance with moves*, which, for input strings S and R is denoted $EDM(S, R)$, and is defined as the minimum number of character edits and *substring (block) moves* to transform one of the strings into the other. A move of block $s[j, k]$ to position h transforms $S = s_1 s_2 \dots s_n$ into $S' = s_1 \dots s_{j-1} s_{k+1} s_{k+2} \dots s_{h-1} s_j \dots s_k s_h \dots s_n$ [4].

If the input strings S and R are permutations of the alphabet σ (so that $|S| = |R| = |\sigma|$) then $EDM(S, R)$ is also called as the transposition distance and is denoted $TED(S, R)$ [1].

Perhaps the most general form of the standard edit distance that involves edit operations on blocks/substrings is the *block edit distance*, denoted $BED(S, R)$. It is defined as the minimum number of single character edits, block moves, as well as *block copies and block uncopies* to transform one of the strings into the other. Copying of a block $s[j, k]$ to position h transforms $S = s_1 s_2 \dots s_n$ into $S' = s_1 \dots s_j s_{j+1} \dots s_k \dots s_{h-1} s_j \dots s_k s_h \dots s_n$. A block uncopy is the inverse of a block copy: it deletes a block $s[j, k]$ provided there exists $s[j', k'] = s[j, k]$ which does not overlap with $s[j, k]$ and transforms S into $S' = s_1 \dots s_{j-1} s_{k+1} \dots s_n$.

Throughout this discussion all edit operations have unit cost and they may overlap; i. e. a character can be edited on multiple times.

Key Results

There are exact and approximate solutions to computing the edit distances described above with varying performance guarantees. As can be expected, the best available running times as well as the approximation factors for computing these edit distances vary considerably with the edit operations allowed.

Exact Computation of the Standard and Permutation Edit Distance

The fastest algorithms for exactly computing the standard edit distance have been available for more than 25 years.

Theorem 1 (Levenshtein [9]) *The standard edit distance $ED(S, R)$ can be computed exactly in time $O(n \cdot m)$ via dynamic programming.*

Theorem 2 (Masek-Paterson [11]) *The standard edit distance $ED(S, R)$ can be computed exactly in time $O(n + n \cdot m / \log^2_{|\sigma|} n)$ via the "four-Russians trick".*

Theorem 3 (Landau-Vishkin [8]) *It is possible to compute $ED(S, R)$ in time $O(n \cdot ED(S, R))$.*

Finally, note that if S and R are permutations of the alphabet σ, $PED(S, R)$ can be computed much faster than the standard edit distance for general strings: Observe that $PED(S,R) = n - LCS(S,R)$ where $LCS(S,R)$ represents the longest common subsequence of S and R. For permutations S, R, $LCS(S,R)$ can be computed in time $O(n \cdot \log\log n)$ [3].

Approximate Computation of the Standard Edit Distance

If some approximation can be tolerated, it is possible to considerably improve the $\tilde{O}(n \cdot m)$ time ($\tilde{O}$ notation hides polylogarithmic factors) available by the techniques above. The fastest algorithm that *approximately* computes the standard edit distance works by *embedding* strings S and R from alphabet σ into shorter strings S' and R' from a larger alphabet σ' [2]. The embedding is achieved by applying a general version of the *Locally Consistent Parsing* [13,14] to partition the strings R and S into *consistent blocks* of size c to $2c-1$; the partitioning is consistent in the sense that identical (long) substrings are partitioned identically. Each block is then replaced with a label such that identical blocks are identically labeled. The resulting strings S' and R' preserve the edit distance between S and R approximately as stated below.

Theorem 4(Batu-Ergun-Sahinalp [2]) *$ED(S,R)$ can be computed in time $\tilde{O}(n^{1+\epsilon})$ within an approximation factor of* $\min\{n^{\frac{1-\epsilon}{3}+o(1)}, (ED(S,R)/n^{\epsilon})^{\frac{1}{2}+o(1)}\}$.

For the case of $\epsilon = 0$, the above result provides an $\tilde{O}(n)$ time algorithm for approximating $ED(S,R)$ within a factor of $\min\{n^{\frac{1}{3}+o(1)}, ED(S,R)^{\frac{1}{2}+o(1)}\}$.

Approximate Computation of Edit Distances Involving Block Edits

For all edit distance variants described above which involve blocks, there are no known polynomial time algorithms; in fact it is NP-hard to compute $TED(S,R)$ [1], $EDM(S,R)$ and $BED(S,R)$ [10]. However, in case S and R are permutations of σ, there are polynomial time algorithms that approximate transposition distance within a constant factor:

Theorem 5 (Bafna-Pevzner [1]) *$TED(S,R)$ can be approximated within a factor of* 1.5 *in $O(n^2)$ time.*

Furthermore, even if S and R are arbitrary strings from σ, it is possible to approximately compute both $BED(S,R)$ and $EDM(S,R)$ in near linear time. More specifically obtain an embedding of S and R to binary vectors $f(S)$ and $f(R)$ such that:

Theorem 6 (Muthukrishnan-Sahinalp [12])
$\frac{||f(S)-f(R)||_1}{\log^* n} \le BED(S,R) \le ||f(S)-f(R)||_1 \cdot \log n$.

In other words, the Hamming distance between $f(S)$ and $f(R)$ approximates $BED(S,R)$ within a factor of $\log n \cdot \log^* n$. Similarly for $EDM(S,R)$, it is possible to embed S and R to integer valued vectors $F(S)$ and $F(R)$ such that:

Theorem 7 (Cormode-Muthukrishnan [4])
$\frac{||F(S)-F(R)||_1}{\log^* n} \le EDM(S,R) \le ||F(S)-F(R)||_1 \cdot \log n$.

In other words, the L_1 distance between $F(S)$ and $F(R)$ approximates $EDM(S,R)$ within a factor of $\log n \cdot \log^* n$.

The embedding of strings S and R into binary vectors $f(S)$ and $f(R)$ is introduced in [5] and is based on the Locally Consistent Parsing described above. To obtain the embedding, one needs to hierarchically partition S and R into growing size *core* blocks. Given an alphabet σ, Locally Consistent Parsing can identify only a limited number of substrings as core blocks. Consider the lexicographic ordering of these core blocks. Each dimension i of the embedding $f(S)$ simply indicates (by setting $f(S)[i] = 1$) whether S includes the ith core block corresponding to the alphabet σ as a substring. Note that if a core block exists in S as a substring, Locally Consistent Parsing will identify it.

Although the embedding above is exponential in size, the resulting binary vector $f(S)$ is very sparse. A simple representation of $f(S)$ and $f(R)$, exploiting their sparseness can be computed in time $O(n \log^* n)$ and the Hamming distance between $f(S)$ and $f(R)$ can be computed in linear time by the use of this representation [12].

The embedding of S and R into integer valued vectors $F(S)$ and $F(R)$ are based on similar techniques. Again, the total time needed to approximate $EDM(S,R)$ within a factor of $\log n \cdot \log^* n$ is $O(n \log^* n)$.

Applications

Edit distances have important uses in computational evolutionary biology, in estimating the evolutionary distance between pairs of genome sequences under various edit operations. There are also several applications to the *document exchange problem* or *document reconciliation problem* where two copies of a text string S have been subject to edit operations (both single character and block edits) by two parties resulting in two versions S_1 and S_2, and the parties communicate to reconcile the differences between the two versions. An information theoretic lower bound on the number of bits to communicate between the two parties is then $\Omega(BED(S,R)) \cdot \log n$. The embedding of S and R to binary strings $f(S)$ and $f(R)$ provides a simple protocol [5] which gives a near-optimal tradeoff between the number of rounds of communication and the total number of bits exchanged and works with high probability.

Another important application is to the Sequence Nearest Neighbors (SNN) problem, which asks to preprocess a set of strings $S_1, \dots, S_k$ so that given an on-line query string R, the string S_i which has the lowest distance of choice to R can be computed in time polynomial with $|R|$ and polylogarithmic with $\sum_{j=1}^{k} |S_j|$. There are no known exact solutions for the SNN problem under any edit distance considered here. However, in [12], the embedding of strings S_i into binary vectors $f(S_i)$, combined with the Approximate Nearest Neighbors results given in [6] for Hamming Distance, provides an approximate solution to the SNN problem under block edit distance as follows.

Theorem 8 (Muthukrishnan-Sahinalp [12]) *It is possible to preprocess a set of strings $S_1, \dots, S_k$ from a given alphabet σ in $O(poly(\sum_{j=1}^{k} |S_j|))$ time such that for any on-line query string R from σ one can compute a string S_i in time $O(polylog(\sum_{j=1}^{k} |S_j|) \cdot poly(|R|))$ which guarantees that for all $h \in [1, k]$, $BED(S_i, R) \leq BED(S_h, R) \cdot \log(\max_j |S_j|) \cdot \log^*(\max_j |S_j|)$.*

Open Problems

It is interesting to note that when dealing with permutations of the alphabet σ the problem of computing both character edit distances and block edit distances become much easier; one can compute $PED(S, R)$ exactly and $TED(S, R)$ within an approximation factor of 1.5 in $\tilde{O}(n)$ time. For arbitrary strings, it is an open question whether one can approximate $TED(S, R)$ or $BED(S, R)$ within a factor of $o(\log n)$ in polynomial time. One recent result in this direction shows that it is not possible to obtain a polylogarithmic approximation to $TED(S, R)$ via a greedy strategy [7]. Furthermore, although there is a lower bound of $\Omega(n^{\frac{1}{3}})$ on the approximation factor that can be achieved for computing the standard edit distance in $\tilde{O}(n)$ time by the use of string embeddings, there is no general lower bound on how closely one can approximate $ED(S, R)$ in near linear time.

Cross References

▶ Sequential Approximate String Matching

Recommended Reading

1. Bafna, V., Pevzner, P.A.: Sorting by Transpositions. SIAM J. Discret. Math. **11**(2), 224–240 (1998)
2. Batu, T., Ergün, F., Sahinalp, S.C.: Oblivious string embeddings and edit distance approximations. Proc. ACM-SIAM SODA 792–801 (2006)
3. Besmaphyatnikh, S., Segal, M.: Enumerating longest increasing subsequences and patience sorting. Inform. Proc. Lett. **76**(1–2), 7–11 (2000)
4. Cormode, G., Muthukrishnan, S.: The string edit distance matching problem with moves. Proc. ACM-SIAM SODA 667–676 (2002)
5. Cormode, G., Paterson, M., Sahinalp, S.C., Vishkin, U.: Communication complexity of document exchange. Proc. ACM-SIAM SODA 197–206 (2000)
6. Indyk, P., Motwani, R.: Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. Proc. ACM STOC 604–613 (1998)
7. Kaplan, H., Shafrir, N.: The greedy algorithm for shortest superstrings. Inform. Proc. Lett. **93**(1), 13–17 (2005)
8. Landau, G., Vishkin, U.: Fast parallel and serial approximate string matching. J. Algorithms **10**, 157–169 (1989)
9. Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions, and reversals. Doklady Akademii Nauk SSSR **163**(4):845–848 (1965) (Russian). Soviet Physics Doklady **10**(8), 707–710 (1966) (English translation)
10. Lopresti, D.P., Tomkins, A.: Block Edit Models for Approximate String Matching. Theoretical. Comput. Sci. **181**(1), 159–179 (1997)
11. Masek, W., Paterson, M.: A faster algorithm for computing string edit distances. J. Comput. Syst. Sci. **20**, 18–31 (1980)
12. Muthukrishnan, S., Sahinalp, S.C.: Approximate nearest neighbors and sequence comparison with block operations. Proc. ACM STOC 416–424 (2000)
13. Sahinalp, S.C., Vishkin, U.: Symmetry breaking for suffix tree construction. ACM STOC 300–309 (1994)
14. Sahinalp, S.C., Vishkin, U.: Efficient Approximate and Dynamic Matching of Patterns Using a Labeling Paradigm. Proc. IEEE FOCS 320–328 (1996)

Efficient Methods for Multiple Sequence Alignment with Guaranteed Error Bounds

1993; Gusfield

FRANCIS CHIN, S. M. YIU
Department of Computer Science,
University of Hong Kong, Hong Kong, China

Keywords and Synonyms

Multiple string alignment; Multiple global alignment

Problem Definition

Multiple sequence alignment is an important problem in computational biology. Applications include finding highly conserved subregions in a given set of biological sequences and inferring the evolutionary history of a set of taxa from their associated biological sequences (e. g., see [6]). There are a number of measures proposed for evaluating the goodness of a multiple alignment, but prior to this work, no efficient methods are known for computing the optimal alignment for any of these measures. The work of Gusfield [5] gives two computationally efficient multiple alignment approximation algorithms for two of the measures with approximation ratio of less than 2. For one of the measures, they also derived a randomized algorithm, which is much faster and with high probability, reports a multiple alignment with small error bounds. To the best knowledge of the entry authors, this work is the first to provide approximation algorithms (with guarantee error bounds) for this problem.

Notations and Definitions

Let X and Y be two strings of alphabet Σ. The pairwise alignment of X and Y maps X and Y into strings X' and Y' that may contain spaces, denoted by '_', where (1) $|X'| = |Y'| = \ell$; and (2) removing spaces from X' and Y' returns X and Y, respectively. The score of the alignment is defined as $d(X', Y') = \sum_{i=1}^{\ell} s(X'(i), Y'(i))$ where $X'(i)$ (and $Y'(i)$) denotes the ith character in X' (and Y') and $s(a, b)$ with $a, b \in \Sigma \cup$ '_' is the distance-based scoring scheme that satisfies the following assumptions.

1. $s('\_', '\_') = 0$;
2. triangular inequality: for any three characters, x, y, z, $s(x, z) \leq s(x, y) + s(y, z)$).

Let $\chi = X_1, X_2, \ldots, X_k$ be a set of $k > 2$ strings of alphabet Σ. A multiple alignment A of these k strings maps $X_1, X_2, \ldots, X_k$ to $X'_1, X'_2, \ldots, X_k$' that may contain spaces such that (1) $|X'_1| = |X'_2| = \cdots = |X'_k| = \ell$; and (2) removing spaces from X_i' returns X_i for all $1 \leq i \leq k$. The multiple alignment A can be represented as a $k \times \ell$ matrix.

The Sum of Pairs (SP) Measure

The score of a multiple alignment A, denoted by SP(A), is defined as the sum of the scores of pairwise alignments induced by A, that is, $\sum_{i<j} d(X'_i, X'_j) =$
$\sum_{i<j} \sum_{p=1}^{\ell} s(X'_i[p], X'_j[p])$ where $1 \leq i < j \leq k$.

Problem 1 *Multiple Sequence Alignment with Minimum SP score*
INPUT: *A set of k strings, a scoring scheme s.*
OUTPUT: *A multiple alignment A of these k strings with minimum SP(A).*

The Tree Alignment (TA) Measure

In this measure, the multiple alignment is derived from an evolutionary tree. For a given set χ of k strings, let $\chi' \supseteq \chi$. An evolutionary tree $T_{\chi'}$ for χ is a tree with at least k nodes, where there is a one-to-one correspondence between the nodes and the strings in χ'. Let $X'_u \in \chi$' be the string for node u. The score of $T_{\chi'}$, denoted by $TA(T_{\chi'})$, is defined as $\sum_{e=(u,v)} D(X'_u, X'_v)$ where e is an edge in $T_{\chi'}$ and $D(X'_u, X'_v)$ denotes the score of the optimal pairwise alignment for X'_u and X'_v. Analogously, the multiple alignment of χ under the TA measure can also be represented by a $|\chi'| \times \ell$ matrix, where $|\chi'| \geq k$, with a score defined as $\sum_{e=(u,v)} d(X'_u, X'_v)$($e$ is an edge in $T_{\chi'}$), similar to the multiple alignment under the SP measure in which the score is the summation of the alignment scores of all pairs of strings. Under the TA measure, since it is always possible to construct the $|\chi'| \times \ell$ matrix such that $d(X'_u, X'_v) = D(X'_u, X'_v)$ for all $e = (u, v)$ in $T_{\chi'}$ and we are usually interested in finding the multiple alignment with the minimum TA value, so $D(X'_u, X'_v)$ is used instead of $d(X'_u, X'_v)$ in the definition of $TA(T_{\chi'})$.

Problem 2 *Multiple Sequence Alignment with Minimum TA score*
INPUT: *A set of k strings, a scoring scheme s.*
OUTPUT: *An evolutionary tree T for these k strings with minimum TA(T).*

Key Results

Theorem 1 *Let A^* be the optimal multiple alignment of the given k strings with minimum SP score. They provide an approximation algorithm (the center star method) that gives a multiple alignment A such that $\frac{SP(A)}{SP(A*)} \leq \frac{2(k-1)}{k} = 2 - \frac{2}{k}$.*

The center star method is to derive a multiple alignment which is consistent with the optimal pairwise alignments of a center string with all the other strings. The bound is derived based on the triangular inequality of the score function. The time complexity of this method is $O(k^2\ell^2)$, where ℓ^2 is the time to solve the pairwise alignment by dynamic programming and k^2 is needed to find the center string, X_c, which gives the minimum value of $\sum_{i \neq c} D(X_c, X_i)$.

Theorem 2 *Let A^* be the optimal multiple alignment of the given k strings with minimum SP score. They provide a randomized algorithm that gives a multiple alignment A such that $\frac{SP(A)}{SP(A*)} \leq 2 + \frac{1}{r-1}$ with probability at least $1 - \left(\frac{r-1}{r}\right)^p$ for any $r > 1$ and $p \geq 1$.*

Instead of computing $\binom{k}{2}$ optimal pairwise alignments to find the best center string, the randomized algorithm only considers p randomly selected strings to be candidates for the best center string, thus this method needs to x compute only $(k-1)p$ optimal pairwise alignments in $O(kp\ell^2)$ time where $1 \leq p \leq k$.

Theorem 3 *Let T^* be the optimal evolutionary tree of the given k strings with minimum TA score. They provide an* *approximation algorithm that gives an evolutionary tree T such that* $\frac{TA(T)}{TA(T*)} \leq \frac{2(k-1)}{k} = 2 - \frac{2}{k}$.

In the algorithm, they first compute all the $\binom{k}{2}$ optimal pairwise alignments to construct a graph with every node representing a distinct string X_i and the weight of each edge (X_i, X_j) as $D(X_i, X_j)$. This step determines the overall time complexity $O(k^2\ell^2)$. Then, they find a minimum spanning tree from the graph. The multiple alignment has to be consistent with the optimal pairwise alignments represented by the edges of this minimum spanning tree.

Applications

Multiple sequence alignment is a fundamental problem in computational biology. In particular, multiple sequence alignment is useful in identifying those common structures, which may only be weakly reflected in the sequence and not easily revealed by pairwise alignment. These common structures may carry important information for their evolutionary history, critical conserved motifs, common 3D molecular structure, as well as biological functions.

More recently, multiple sequence alignment is also used in revealing non-coding RNAs (ncRNAs) [3]. In this type of multiple alignment, we are not only align the underlying sequences, but also the secondary structures (refer to chap. 16 of [10] for a brief introduction of secondary structure of a RNA) of the RNAs. Researchers believe that ncRNAs that belong to the same family should have common components giving a similar secondary structure. The multiple alignment can help to locate and identify these common components.

Open Problems

A number of open problems related to the work of Gusfield remain open. For the SP measure, the center star method can be extended to the q-star method ($q > 2$) with approximation ratio of $2 - q/k$ ([1,7], sect. 7.5 of [8]). Whether there exists an approximation algorithm with better approximation ratio or with better time complexity is still unknown. For the TA measure, to be the best knowledge of the entry authors, the approximation ratio in Theorem 3 is currently the best result.

Another interesting direction related to this problem is the constrained multiple sequence alignment problem [9] which requires the multiple alignment to contain certain aligned characters with respect to a given constrained sequence. The best known result [2] is an approximation algorithm (also follows the idea of center star method) which gives an alignment with approximation ratio of $2 - 2/k$ for k strings.

For the complexity of the problem, Wang and Jiang [11] were the first to prove the NP-hardness of the problem with SP score under a *non-metric* distance measure over a 4 symbol alphabet. More recently, in [4], the multiple alignment problem with SP score, star alignment, and TA score have been proved to be NP-hard for all binary or larger alphabets under *any metric*. Developing efficient approximation algorithms with good bounds for any of these measures is desirable.

Experimental Results

Two experiments have been reported in the paper showing that the worst case error bounds in Theorems 1 and 2 (for the SP measure) are pessimistic compared to the typical situation arising in practice.

The scoring scheme used in the experiments is: $s(a, b) = 0$ if $a = b$; $s(a, b) = 1$ if either a or b is a space; otherwise $s(a, b) = 2$. Since computing the optimal multiple alignment with minimum SP score has been shown to be NP-hard, they evaluate the performance of their algorithms using the lower bound of $\sum_{i<j} D(X_i, X_j)$ (recall that $D(X_i, X_j)$ is the score of the optimal pairwise alignment of X_i and X_j). They have aligned 19 similar amino acid sequences with average length of 60 of homeoboxs from different species. The ratio of the scores of reported alignment by the center star method to the lower bound is only 1.018 which is far from the worst case error bound given in Theorem 1. They also aligned 10 not-so-similar sequences near the homeoboxs, the ratio of the reported alignment to the lower bound is 1.162. Results also show that the alignment obtained by the randomized algorithm is usually not far away from the lower bound.

Data Sets

The exact sequences used in the experiments are not provided.

Cross References

▶ Statistical Multiple Alignment

Recommended Reading

1. Bafna, V., Lawler, E.L., Pevzner, P.A.: Approximation algorithms for multiple sequence alignment. Theor. Comput. Sci. **182**, 233–244 (1997)
2. Francis, Y.L., Chin, N.L.H., Lam, T.W., Prudence, W.H.W.: Efficient constrained multiple sequence alignment with performance guarantee. J. Bioinform. Comput. Biol. **3**(1), 1–18 (2005)
3. Dalli, D., Wilm, A., Mainz, I., Stegar, G.: STRAL: progressive alignment of non-coding RNA using base pairing probability vectors in quadratic time. Bioinformatics **22**(13), 1593–1599 (2006)

4. Elias, I.: Setting the intractability of multiple alignment. In: Proc. of the 14th Annual International Symposium on Algorithms and Computation (ISAAC 2003), 2003, pp. 352–363
5. Gusfield, D.: Efficient methods for multiple sequence alignment with guaranteed error bounds. Bull. Math. Biol. **55**(1), 141–154 (1993)
6. Pevsner, J.: Bioinformatics and functional genomics. Wiley, New York (2003)
7. Pevzner, P.A.: Multiple alignment, communication cost, and graph matching. SIAM J. Appl. Math. **52**, 1763–1779 (1992)
8. Pevzner, P.A.: Computational molecular biology: an algorithmic approach. MIT Press, Cambridge, MA (2000)
9. Tang, C.Y., Lu, C.L., Chang, M.D.T., Tsai, Y.T., Sun, Y.J., Chao, K.M., Chang, J.M., Chiou, Y.H., Wu, C.M., Chang, H.T., Chou, W.I.: Constrained multiple sequence alignment tool development and its application to RNase family alignment. In: Proc. of the First IEEE Computer Society Bioinformatics Conference (CSB 2002), 2002, pp. 127–137
10. Tompa, M.: Lecture notes. Department of Computer Science & Engineering, University of Washington. http://www.cs.washington.edu/education/courses/527/00wi/. (2000)
11. Wang, L. Jiang, T.: On the complexity of multiple sequence alignment. J. Comp. Biol. **1**, 337–48 (1994)

Engineering Algorithms for Computational Biology

2002; Bader, Moret, Warnow

David A. Bader
College of Computing, Georgia Institute of Technology, Atlanta, GA, USA

Keywords and Synonyms

High-performance computational biology

Problem Definition

In the 50 years since the discovery of the structure of DNA, and with new techniques for sequencing the entire genome of organisms, biology is rapidly moving towards a data-intensive, computational science. Many of the newly faced challenges require high-performance computing, either due to the massive-parallelism required by the problem, or the difficult optimization problems that are often combinatoric and NP-hard. Unlike the traditional uses of supercomputers for regular, numerical computing, many problems in biology are irregular in structure, significantly more challenging to parallelize, and integer-based using abstract data structures.

Biologists are in search of biomolecular sequence data, for its comparison with other genomes, and because its structure determines function and leads to the understanding of biochemical pathways, disease prevention and cure, and the mechanisms of life itself. Computational biology has been aided by recent advances in both technology and algorithms; for instance, the ability to sequence short contiguous strings of DNA and from these reconstruct the whole genome and the proliferation of high-speed microarray, gene, and protein chips for the study of gene expression and function determination. These high-throughput techniques have led to an exponential growth of available genomic data.

Algorithms for solving problems from computational biology often require parallel processing techniques due to the data- and compute-intensive nature of the computations. Many problems use polynomial time algorithms (e. g., all-to-all comparisons) but have long running times due to the large number of items in the input; for example, the assembly of an entire genome or the all-to-all comparison of gene sequence data. Other problems are compute-intensive due to their inherent algorithmic complexity, such as protein folding and reconstructing evolutionary histories from molecular data, that are known to be NP-hard (or harder) and often require approximations that are also complex.

Key Results

None

Applications

Phylogeny Reconstruction: A phylogeny is a representation of the evolutionary history of a collection of organisms or genes (known as taxa). The basic assumption of process necessary to phylogenetic reconstruction is repeated divergence within species or genes. A phylogenetic reconstruction is usually depicted as a tree, in which modern taxa are depicted at the leaves and ancestral taxa occupy internal nodes, with the edges of the tree denoting evolutionary relationships among the taxa. Reconstructing phylogenies is a major component of modern research programs in biology and medicine (as well as linguistics). Naturally, scientists are interested in phylogenies for the sake of knowledge, but such analyses also have many uses in applied research and in the commercial arena. Existing phylogenetic reconstruction techniques suffer from serious problems of running time (or, when fast, of accuracy). The problem is particularly serious for large data sets: even though data sets comprised of sequence from a single gene continue to pose challenges (e. g., some analyses are still running after two years of computation on medium-sized clusters), using whole-genome data (such as gene content and gene order) gives rise to even more formidable computational problems, particularly in data sets with large numbers of genes and highly-rearranged genomes.

To date, almost every model of speciation and genomic evolution used in phylogenetic reconstruction has given rise to NP-hard optimization problems. Three major classes of methods are in common use. Heuristics (a natural consequence of the NP-hardness of the problems) run quickly, but may offer no quality guarantees and may not even have a well-defined optimization criterion, such as the popular *neighbor-joining* heuristic [9]. Optimization based on the criterion of *maximum parsimony* (MP) [4] seeks the phylogeny with the least total amount of change needed to explain modern data. Finally, optimization based on the criterion of *maximum likelihood* (ML) [5] seeks the phylogeny that is the most likely to have given rise to the modern data.

Heuristics are fast and often rival the optimization methods in terms of accuracy, at least on datasets of moderate size. Parsimony-based methods may take exponential time, but, at least for DNA and amino acid data, can often be run to completion on datasets of moderate size. Methods based on maximum likelihood are very slow (the point estimation problem alone appears intractable) and thus restricted to very small instances, and also require many more assumptions than parsimony-based methods, but appear capable of outperforming the others in terms of the quality of solutions when these assumptions are met. Both MP- and ML-based analyses are often run with various heuristics to ensure timely termination of the computation, with mostly unquantified effects on the quality of the answers returned.

Thus there is ample scope for the application of high-performance algorithm engineering in the area. As in all scientific computing areas, biologists want to study a particular dataset and are willing to spend months and even years in the process: accurate branch prediction is the main goal. However, since all exact algorithms scale exponentially (or worse, in the case of ML approaches) with the number of taxa, speed remains a crucial parameter – otherwise few datasets of more than a few dozen taxa could ever be analyzed.

Experimental Results

As an illustration, this entry briefly describes a high-performance software suite, GRAPPA (Genome Rearrangement Analysis through Parsimony and other Phylogenetic Algorithms) developed by Bader et al. *GRAPPA* extends Sankoff and Blanchette's breakpoint phylogeny algorithm [10] into the more biologically-meaningful inversion phylogeny and provides a highly-optimized code that can make use of distributed- and shared-memory parallel systems (see [1,2,6,7,8,11] for details). In [3], Bader et al. gives the first linear-time algorithm and fast implementation for computing inversion distance between two signed permutations. *GRAPPA* was run on a 512-processor IBM Linux cluster with Myrinet and obtained a 512-fold speedup (linear speedup with respect to the number of processors): a complete breakpoint analysis (with the more demanding inversion distance used in lieu of breakpoint distance) for the 13 genomes in the Campanulaceae data set ran in less than 1.5 hours in an October 2000 run, for a *million-fold* speedup over the original implementation. The latest version features significantly improved bounds and new distance correction methods and, on the same dataset, exhibits a speedup factor of *over one billion*. GRAPPA achieves this speedup through a combination of parallelism and high-performance algorithm engineering. Although such spectacular speedups will not always be realized, many algorithmic approaches now in use in the biological, pharmaceutical, and medical communities may benefit tremendously from such an application of high-performance techniques and platforms.

This example indicates the potential of applying high-performance algorithm engineering techniques to applications in computational biology, especially in areas that involve complex optimizations: Bader's reimplementation did not require new algorithms or entirely new techniques, yet achieved gains that turned an impractical approach into a usable one.

Cross References

- ▶ Distance-Based Phylogeny Reconstruction (Fast-Converging)
- ▶ Distance-Based Phylogeny Reconstruction (Optimal Radius)
- ▶ Efficient Methods for Multiple Sequence Alignment with Guaranteed Error Bounds
- ▶ High Performance Algorithm Engineering for Large-scale Problems
- ▶ Local Alignment (with Affine Gap Weights)
- ▶ Local Alignment (with Concave Gap Weights)
- ▶ Multiplex PCR for Gap Closing (Whole-genome Assembly)
- ▶ Peptide De Novo Sequencing with MS/MS
- ▶ Perfect Phylogeny Haplotyping
- ▶ Phylogenetic Tree Construction from a Distance Matrix
- ▶ Phylogeny Reconstruction
- ▶ Sorting Signed Permutations by Reversal (Reversal Distance)
- ▶ Sorting Signed Permutations by Reversal (Reversal Sequence)

► Sorting by Transpositions and Reversals (Approx Ratio 1.5)
► Substring Parsimony

Recommended Reading

1. Bader, D.A., Moret, B.M.E., Warnow, T., Wyman, S.K., Yan, M.: High-performance algorithm engineering for gene-order phylogenies. In: DIMACS Workshop on Whole Genome Comparison, Rutgers University, Piscataway, NJ (2001)
2. Bader, D.A., Moret, B.M.E., Vawter, L.: Industrial applications of high-performance computing for phylogeny reconstruction. In: Siegel, H.J. (ed.) Proc. SPIE Commercial Applications for High-Performance Computing, vol. 4528, pp. 159–168, Denver, CO (2001)
3. Bader, D.A., Moret, B.M.E., Yan, M.: A linear-time algorithm for computing inversion distance between signed permutations with an experimental study. J. Comp. Biol. **8**(5), 483–491 (2001)
4. Farris, J.S.: The logical basis of phylogenetic analysis. In: Platnick, N.I., Funk, V.A. (eds.) Advances in Cladistics, pp. 1–36. Columbia Univ. Press, New York (1983)
5. Felsenstein, J.: Evolutionary trees from DNA sequences: a maximum likelihood approach. J. Mol. Evol. **17**, 368–376 (1981)
6. Moret, B.M.E., Bader, D.A., Warnow, T., Wyman, S.K., Yan, M.: GRAPPA: a highperformance computational tool for phylogeny reconstruction from gene-order data. In: Proc. Botany, Albuquerque, August 2001
7. Moret, B.M.E., Bader, D.A., Warnow, T.: High-performance algorithm engineering for computational phylogenetics. J. Supercomp. **22**, 99–111 (2002) Special issue on the best papers from ICCS'01
8. Moret, B.M.E., Wyman, S., Bader, D.A., Warnow, T., Yan, M.: A new implementation and detailed study of breakpoint analysis. In: Proc. 6th Pacific Symp. Biocomputing (PSB 2001), pp. 583–594, Hawaii, January 2001
9. Saitou, N., Nei, M.: The neighbor-joining method: A new method for reconstruction of phylogenetic trees. Mol. Biol. Evol. **4**, 406–425 (1987)
10. Sankoff, D., Blanchette, M.: Multiple genome rearrangement and breakpoint phylogeny. J. Comp. Biol. **5**, 555–570 (1998)
11. Yan, M.: High Performance Algorithms for Phylogeny Reconstruction with Maximum Parsimony. Ph. D. thesis, Electrical and Computer Engineering Department, University of New Mexico, Albuquerque, January 2004

Engineering Algorithms for Large Network Applications

2002; Schulz, Wagner, Zaroliagis

CHRISTOS ZAROLIAGIS
Department of Computer Engineering & Informatics, University of Patras, Patras, Greece

Problem Definition

Dealing effectively with applications in large networks, it typically requires the efficient solution of one ore more underlying algorithmic problems. Due to the size of the network, a considerable effort is inevitable in order to achieve the desired efficiency in the algorithm.

One of the primary tasks in large network applications is to answer queries for finding best routes or paths as efficiently as possible. Quite often, the challenge is to process a vast number of such queries on-line: a typical situation encountered in several real-time applications (e. g., traffic information systems, public transportation systems) concerns a query-intensive scenario, where a central server has to answer a huge number of on-line customer queries asking for their best routes (or optimal itineraries). The main goal in such an application is to reduce the (average) response time for a query.

Answering a best route (or optimal itinerary) query translates in computing a minimum cost (shortest) path on a suitably defined directed graph (digraph) with non-negative edge costs. This in turn implies that the core algorithmic problem underlying the efficient answering of queries is the single-source single-target shortest path problem.

Although the straightforward approach of pre-computing and storing shortest paths for all pairs of vertices would enabling the optimal answering of shortest path queries, the quadratic space requirements for digraphs with more than 10^5 vertices makes such an approach prohibitive for large and very large networks. For this reason, the main goal of almost all known approaches is to keep the space requirements as small as possible. This in turn implies that one can afford a heavy (in time) preprocessing, which does not blow up space, in order to speed-up the query time.

The most commonly used approach for answering shortest path queries employs Dijkstra's algorithm and/or variants of it. Consequently, the main challenge is how to reduce the algorithm's *search-space* (number of vertices visited), as this would immediately yield a better query time.

Key Results

All results discussed concern answering of *optimal* (or *exact* or *distance-preserving*) shortest paths under the aforementioned query-intensive scenario, and are all based on the following generic approach. A preprocessing of the input network $G = (V, E)$ takes place that results in a data structure of size $O(|V| + |E|)$ (i. e., linear to the size of G). The data structure contains additional information regarding certain shortest paths that can be used later during querying.

Depending on the pre-computed additional information as well as on the way a shortest path query is answered, two approaches can be distinguished. In the first approach, *graph annotation*, the additional information is attached to vertices or edges of the graph. Then, speed-up techniques to Dijkstra's algorithm are employed that, based on this information, decide quickly which part of the graph does not need to be searched. In the second approach, an *auxiliary graph* G' is constructed hierarchically. A shortest path query is then answered by searching only a small part of G', using Dijkstra's algorithm enhanced with heuristics to further speed-up the query time.

In the following, the key results of the first [3,4,9,11] and the second approach [1,2,5,7,8,10] are discussed, as well as results concerning modeling issues.

First Approach – Graph Annotation

The first work under this approach concerns the study in [9] on large railway networks. In that paper, two new heuristics are introduced: the *angle-restriction* (that tries to reduce the search space by taking advantage of the geometric layout of the vertices) and the *selection of stations* (a subset of vertices is selected among which all pairs shortest paths are pre-computed). These two heuristics along with a combination of the classical *goal-directed* or *A^* search* turned out to be rather efficient. Moreover, they motivated two important generalizations [10,11] that gave further improvements to shortest path query times.

The full exploitation of geometry-based heuristics was investigated in [11], where both street and railway networks are considered. In that paper, it is shown that the search space of Dijkstra's algorithm can be significantly reduced (to 5%–10% of the initial graph size) by extracting geometric information from a given layout of the graph and by encapsulating pre-computed shortest path information in resulted geometric objects, called *containers*. Moreover, the dynamic case of the problem was investigated, where edge costs are subject to change and the geometric containers have to be updated.

A powerful modification to the classical Dijkstra's algorithm, called *reach-based routing*, was presented in [4]. Every vertex is assigned a so-called *reach value* that determines whether a particular vertex will be considered during Dijkstra's algorithm. A vertex is excluded from consideration if its reach value is small; that is, if it does not contribute to any path long enough to be of use for the current query.

A considerable enhancement of the classical *A^* search* algorithm using landmarks (selected vertices like in [9,10]) and the triangle inequality with respect to the shortest path distances was shown in [3]. Landmarks and triangle inequality help to provide better lower bounds and hence boost A^* search.

Second Approach – Auxiliary Graph

The first work under this approach concerns the study in [10], where a new hierarchical decomposition technique is introduced called *multi-level graph*. A multi-level graph $\mathcal{M}$ is a digraph which is determined by a sequence of subsets of V and which extends E by adding multiple levels of edges. This allows to efficiently construct, during querying, a subgraph of $\mathcal{M}$ which is substantially smaller than G and in which the shortest path distance between any of its vertices is equal to the shortest path distance between the same vertices in G. Further improvements of this approach have been presented recently in [1]. A refinement of the above idea was introduced in [5], where the multi-level overlay graphs are introduced. In such a graph, the decomposition hierarchy is not determined by application-specific information as it happens in [9,10].

An alternative hierarchical decomposition technique, called *highway hierarchies*, was presented in [7]. The approach takes advantage of the inherent hierarchy possessed by real-world road networks and computes a hierarchy of coarser views of the input graph. Then, the shortest path query algorithm considers mainly the (much smaller in size) coarser views, thus achieving dramatic speed-ups in query time. A revision and improvement of this method was given in [8]. A powerful combination of the highway hierarchies with the ideas in [3] was reported in [2].

Modeling Issues

The modeling of the original best route (or optimal itinerary) problem on a large network to a shortest path problem in a suitably defined directed graph with appropriate edge costs also plays a significant role in reducing the query time. Modeling issues are thoroughly investigated in [6]. In that paper, the first experimental comparison of two important approaches (time-expanded versus time-dependent) is carried out, along with new extensions of them towards realistic modeling. In addition, several new heuristics are introduced to speed-up query time.

Applications

Answering shortest path queries in large graphs has a multitude of applications, especially in traffic information systems under the aforementioned scenario; that is, a central server has to answer, as fast as possible, a huge number of on-line customer queries asking for their best routes or itineraries. Other applications of the above scenario involve route planning systems for cars, bikes and hikers, public transport systems for itinerary information of scheduled vehicles (like trains or buses), answering queries in spatial databases, and web searching. All the above applications concern real-time systems in which users continuously enter their requests for finding their best connections or routes. Hence, the main goal is to reduce the (average) response time for answering a query.

Open Problems

Real-world networks increase constantly in size either as a result of accumulation of more and more information on them, or as a result of the digital convergence of media services, communication networks, and devices. This scaling-up of networks makes the scalability of the underlying algorithms questionable. As the networks continue to grow, there will be a constant need for designing faster algorithms to support core algorithmic problems.

Experimental Results

All papers discussed in Sect. "Key Results" contain important experimental studies on the various techniques they investigate.

Data Sets

The data sets used in [6,11] are available from http://lso-compendium.cti.gr/ under problems 26 and 20, respectively.

The data sets used in [1,2] are available from http://www.dis.uniroma1.it/~challenge9/.

URL to Code

The code used in [9] is available from http://doi.acm.org/10.1145/351827.384254.

The code used in [6,11] is available from http://lso-compendium.cti.gr/ under problems 26 and 20, respectively.

The code used in [3] is available from http://www.avglab.com/andrew/soft.html.

Cross References

► Implementation Challenge for Shortest Paths
► Shortest Paths Approaches for Timetable Information

Recommended Reading

1. Delling, D., Holzer, M., Müller, K., Schulz, F., Wagner, D.: High-Performance Multi-Level Graphs. In: 9th DIMACS Challenge on Shortest Paths, Nov 2006. Rutgers University, USA (2006)
2. Delling, D., Sanders, P., Schultes, D., Wagner, D.: Highway Hierarchies Star. In: 9th DIMACS Challenge on Shortest Paths, Nov 2006 Rutgers University, USA (2006)
3. Goldberg, A.V., Harrelson, C.: Computing the Shortest Path: A^* Search Meets Graph Theory. In: Proc. 16th ACM-SIAM Symposium on Discrete Algorithms – SODA, pp. 156–165. ACM, New York and SIAM, Philadelphia (2005)
4. Gutman, R.: Reach-based Routing: A New Approach to Shortest Path Algorithms Optimized for Road Networks. In: Algorithm Engineering and Experiments – ALENEX (SIAM, 2004), pp. 100–111. SIAM, Philadelphia (2004)
5. Holzer, M., Schulz, F., Wagner, D.: Engineering Multi-Level Overlay Graphs for Shortest-Path Queries. In: Algorithm Engineering and Experiments – ALENEX (SIAM, 2006), pp. 156–170. SIAM, Philadelphia (2006)
6. Pyrga, E., Schulz, F., Wagner, D., Zaroliagis, C.: Efficient Models for Timetable Information in Public Transportation Systems. ACM J. Exp. Algorithmic **12**(2.4), 1–39 (2007)
7. Sanders, P., Schultes, D.: Highway Hierarchies Hasten Exact Shortest Path Queries. In: Algorithms – ESA 2005. Lect. Note Comp. Sci. **3669**, 568–579 (2005)
8. Sanders, P., Schultes, D.: Engineering Highway Hierarchies. In: Algorithms – ESA 2006. Lect. Note Comp. Sci. **4168**, 804–816 (2006)
9. Schulz, F., Wagner, D., Weihe, K.: Dijkstra's Algorithm On-Line: An Empirical Case Study from Public Railroad Transport. ACM J. Exp. Algorithmics **5**(12), 1–23 (2000)
10. Schulz, F., Wagner, D., Zaroliagis, C.: Using Multi-Level Graphs for Timetable Information in Railway Systems. In: Algorithm Engineering and Experiments – ALENEX 2002. Lect. Note Comp. Sci. **2409**, 43–59 (2002)
11. Wagner, D., Willhalm, T., Zaroliagis, C.: Geometric Containers for Efficient Shortest Path Computation. ACM J. Exp. Algorithmics **10**(1.3), 1–30 (2005)

Engineering Geometric Algorithms

2004; Halperin

DAN HALPERIN
School of Computer Science,
Tel-Aviv University, Tel Aviv, Israel

Keywords and Synonyms

Certified and efficient implementation of geometric algorithms; Geometric computing with certified numerics and topology

Problem Definition

Transforming a theoretical geometric algorithm into an effective computer program abounds with hurdles. Overcoming these difficulties is the concern of *engineering geometric algorithms*, which deals, more generally, with the design and implementation of certified and efficient solutions to algorithmic problems of geometric nature. Typical problems in this family include the construction of Voronoi diagrams, triangulations, arrangements of curves and surfaces (namely, space subdivisions), two- or higher-dimensional search structures, convex hulls and more.

Geometric algorithms strongly couple topological/combinatorial structures (e. g., a graph describing the triangulation of a set of points) on the one hand, with numerical information (e. g., the coordinates of the vertices of the triangulation) on the other. Slight errors in the numerical calculations, which in many areas of science and engineering can be tolerated, may lead to detrimental mistakes in the topological structure, causing the computer program to crash, to loop infinitely, or plainly to give wrong results.

Straightforward implementation of geometric algorithms as they appear in a textbook, using standard machine arithmetic, is most likely to fail. This entry is concerned only with *certified* solutions, namely, solutions that are guaranteed to construct the exact desired structure or a good approximation of it; such solutions are often referred to as *robust*.

The goal of engineering geometric algorithms can be restated as follows: *Design and implement geometric algorithms that are at once robust and efficient in practice*.

Much of the difficulty in adapting in practice the existing vast algorithmic literature in computational geometry comes from the assumptions that are typically made in the theoretical study of geometric algorithms that (1) the input is in general position, namely, degenerate input is precluded, (2) computation is performed on an ideal computer that can carry out real arithmetic to infinite precision (so-called real RAM), and (3) the cost of operating on a small number of simple geometric objects is "unit" time (e. g., equal cost is assigned to intersecting three spheres and to comparing two integer numbers).

Now, in real life, geometric input is quite often degenerate, machine precision is limited, and operations on a small number of simple geometric objects within the same algorithm may differ hundredfold and more in the time they take to execute (when aiming for certified results). Just implementing an algorithm carefully may not suffice and often redesign is called for.

Key Results

Tremendous efforts have been invested in the design and implementation of robust computational-geometry software in recent years. Two notable large-scale efforts are the CGAL library [1] and the geometric part of the LEDA library [14]. These are jointly reviewed in the survey by Kettner and Näher [13]. Numerous other relevant projects, which for space constraints are not reviewed here, are surveyed by Joswig [12] with extensive references to papers and Web sites.

A fundamental engineering decision to take when coming to implement a geometric algorithm is what will the underlying arithmetic be, that is, whether to opt for exact computation or use the machine floating-point arithmetic. (Other less commonly used options exist as well.) To date, the CGAL and LEDA libraries are almost exclusively based on exact computation. One of the reasons for this exclusivity is that exact computation emulates the ideal computer (for restricted problems) and makes the adaptation of algorithms from theory to software easier. This is facilitated by major headway in developing tools for efficient computation with rational or algebraic numbers (GMP [3], LEDA [14], CORE [2] and more). On top of these tools, clever techniques for reducing the amount of exact computation were developed, such as floating-point filters and the higher- level geometric filtering.

The alternative is to use the machine floating-point arithmetic, having the advantage of being very fast. However, it is nowhere near the ideal infinite precision arithmetic assumed in the theoretical study of geometric algorithms and algorithms have to be carefully redesigned. See, for example, the discussion about imprecision in the manual of QHULL, the convex hull program by Barber et al. [5]. Over the years a variety of specially tailored floating-point variants of algorithms have been proposed, for example, the carefully crafted VRONI package by Held [11], which computes the Voronoi diagram of points and line segments using standard floating-point arithmetic, based on the topology-oriented approach of Sugihara and Iri. While VRONI works very well in practice, it is not theoretically certified. *Controlled perturbation* [9] emerges as a systematic method to produce certified approximations of complex geometric constructs while using floating-point arithmetic: the input is perturbed such that all predicates are computed accurately even with the limited-precision machine arithmetic, and a method is given to bound the necessary magnitude of perturbation that will guarantee the successful completion of the computation.

Another decision to take is how to represent the output of the algorithm, where the major issue is typically how to represent the coordinates of vertices of the output structure(s). Interestingly, this question is crucial when using exact computation since there the output coordinates can be prohibitively large or simply impossible to finitely enumerate. (One should note though that many geometric algorithms are *selective* only, namely, they do not produce new geometric entities but just select and order subsets of the input coordinates. For example, the output of an algorithm for computing the convex hull of a set of points in the plane is an ordering of a subset of the input points. No new point is computed. The discussion in this paragraph mostly applies to algorithms that output new geometric constructs, such as the intersection point of two lines.) But even when using floating-point arithmetic, one may prefer to have a more compact bit-size representation than, say, machine doubles. In this direction there is an effective, well-studied solution for the case of polygonal objects in the plane, called *snap rounding*, where vertices and intersection points are snapped to grid vertices while retaining certain topological properties of the exact desired structure. Rounding with guarantees is in general a very difficult problem, and already for polyhedral objects in 3-space the current attempts at generalizing snap rounding are very costly (increasing the complexity of the rounded objects to the third, or even higher, power).

Then there are a variety of engineering issues depending on the problem at hand. Following are two examples of engineering studies where the experience in practice is different from what the asymptotic resource measures imply. The examples relate to fundamental steps in many geometric algorithms: decomposition and point location.

Decomposition

A basic step in many geometric algorithms is to decompose a (possibly complex) geometric object into simpler subobjects, where each subobject typically has constant descriptive complexity. A well-known example is the triangulation of a polygon. The choice of decomposition may have a significant effect on the efficiency in practice of various algorithms that rely on decomposition. Such is the case when constructing Minkowski sums of polygons in the plane. The Minkowski sum of two sets A and B in $\mathbb{R}^d$ is the vector sum of the two sets $A \oplus B = \{a + b | a \in A, b \in B\}$. The simplest approach to computing Minkowski sums of two polygons in the plane proceeds in three steps: triangulate each polygon, then compute the sum of each triangle of one polygon with each triangle of the other, and finally take the union of all the subsums. In asymptotic measures, the choice of triangulation (over alternative decompositions) has no effect. In practice though, triangulation is probably the worst choice compared with other convex decompositions, even fairly simple heuristic ones (not necessarily optimal), as shown by experiments on a dozen different decomposition methods [4]. The explanation is that triangulation increases the overall complexity of the subsums and in turn makes the union stage more complex––indeed by a constant factor, but a noticeable factor in practice. Similar phenomena were observed in other situations as well. For example, when using the prevalent vertical decomposition of arrangements––often it is too costly compared with sparser decompositions (i. e., decompositions that add fewer extra features).

Point Location

A recurring problem in geometric computing is to process given planar subdivision (planar map), so as to efficiently answer *point-location* queries: Given a point q in the plane, which face of the map contains q? Over the years a variety of point-location algorithms for planar maps were implemented in CGAL, in particular, a hierarchical search structure that guarantees logarithmic query time after expected $O(n \log n)$ preprocessing time of a map with n edges. This algorithm is referred to in CGAL as the *RIC* point-location algorithm after the preprocessing method which uses randomized incremental construction. Several simpler, easier-to-program algorithms for point location were also implemented. None of the latter beats the RIC algorithm in query time. However, the RIC is by far the slowest of all the implemented algorithms in terms of preprocessing, which in many scenarios renders it less effective. One of the simpler methods devised is a variant of the well-known *jump-and-walk* approach to point location. The algorithm scatters points (so-called *landmarks*) in the map and maintains the landmarks (together with their containing faces) in a nearest-neighbor search structure. Once a query q is issued it finds the nearest landmark ℓ to q, and "walks" in the map from ℓ toward q along the straight line segment connecting them. This landmark approach offers query time that is only slightly more expensive than the RIC method while being very efficient in preprocessing. The full details can be found in [10]. This is yet another consideration when designing (geometric) algorithms: the cost of preprocessing (and storage) versus the cost of a query. Quite often the effective (practical) tradeoff between these costs needs to be deduced experimentally.

Applications

Geometric algorithms are useful in many areas. Triangulations and arrangements are examples of basic constructs that have been intensively studied in computational geometry, carefully implemented and experimented with, as well as used in diverse applications.

Triangulations

Triangulations in two and three dimensions are implemented in CGAL [7]. In fact, CGAL offers many variants of triangulations useful for different applications. Among the applications where CGAL triangulations are employed are meshing, molecular modeling, meteorology, photogrammetry, and geographic information systems (GIS). For other available triangulation packages, see the survey by Joswig [12].

Arrangements

Arrangements of curves in the plane are supported by CGAL [15], as well as envelopes of surfaces in three-dimensional space. Forthcoming is support also for arrangements of curves on surfaces. CGAL arrangements have been used in motion planning algorithms, computer-aided design and manufacturing, GIS, computer graphics, and more (see Chap. 1 in [6]).

Open Problems

In spite of the significant progress in certified implementation of effective geometric algorithms, the existing theoretical algorithmic solutions for many problems still need adaptation or redesign to be useful in practice. One example where progress can have wide repercussions is devising effective decompositions for curved geometric objects (e. g., arrangements) in the plane and for higher-dimensional objects. As mentioned earlier, suitable decompositions can have a significant effect on the performance of geometric algorithms in practice.

Certified fixed-precision geometric computing lags behind the exact computing paradigm in terms of available robust software, and moving forward in this direction is a major challenge. For example, creating a certified floating-point counterpart to CGAL is a desirable (and highly intricate) task.

Another important tool that is largely missing is consistent and efficient rounding of geometric objects. As mentioned earlier, a fairly satisfactory solution exists for polygonal objects in the plane. Good techniques are missing for curved objects in the plane and for higher-dimensional objects (both linear and curved).

URL to Code

http://www.cgal.org

Cross References

▶ LEDA: a Library of Efficient Algorithms
▶ Robust Geometric Computation

Recommended Reading

Conferences publishing papers on the topic include the ACM Symposium on Computational Geometry (SoCG), the Workshop on Algorithm Engineering and Experiments (ALENEX), the Engineering and Applications Track of the European Symposium on Algorithms (ESA), its predecessor and the Workshop on Experimental Algorithms (WEA). Relevant journals include the *ACM Journal on Experimental Algorithmics, Computational Geometry: Theory and Applications* and the *International Journal of Computational Geometry and Applications*. A wide range of relevant aspects are discussed in the recent book edited by Boissonnat and Teillaud [6], titled *Effective Computational Geometry for Curves and Surfaces*.

1. The CGAL project homepage. http://www.cgal.org/. Accessed 6 Apr 2008
2. The CORE library homepage. http://www.cs.nyu.edu/exact/core/. Accessed 6 Apr 2008
3. The GMP webpage. http://gmplib.org/. Accessed 6 Apr 2008
4. Agarwal, P.K., Flato, E., Halperin, D.: Polygon decomposition for efficient construction of Minkowski sums. Comput. Geom. Theor. Appl. **21**(1–2), 39–61 (2002)
5. Barber, C.B., Dobkin, D.P., Huhdanpaa, H.T.: Imprecision in QHULL. http://www.qhull.org/html/qh-impre.htm. Accessed 6 Apr 2008
6. Boissonnat, J.-D., Teillaud, M. (eds.) Effective Computational Geometry for Curves and Surfaces. Springer, Berlin (2006)
7. Boissonat, J.-D., Devillers, O., Pion, S., Teillaud, M., Yvinec, M.: Triangulations in CGAL. Comput. Geom. Theor. Appl. **22**(1–3), 5-19 (2002)
8. Fabri, A., Giezeman, G.-J., Kettner, L., Schirra, S., Schönherr, S.: On the design of CGAL a computational geometry algorithms library. Softw. Pract. Experience **30**(11), 1167–1202 (2000)
9. Halperin, D., Leiserowitz, E.: Controlled perturbation for arrangements of circles. Int. J. Comput. Geom. Appl. **14**(4–5), 277–310 (2004)
10. Haran, I., Halperin, D.: An experimental study of point location in general planar arrangements. In: Proceedings of 8th Workshop on Algorithm Engineering and Experiments, pp. 16–25 (2006)
11. Held, M.: VRONI: An engineering approach to the reliable and efficient computation of Voronoi diagrams of points and line segments. Comput. Geom. Theor. Appl. **18**(2), 95–123 (2001)
12. Joswig, M.: Software. In: Goodman, J.E., O'Rourke, J. (eds.) Handbook of Discrete and Computational Geometry, 2nd edn., chap. 64, pp. 1415–1433. Chapman & Hall/CRC, Boca Raton (2004)
13. Kettner, L., Näher, S.: Two computational geometry libraries: LEDA and CGAL. In: Goodman, J.E., O'Rourke, J. (eds.) Handbook of Discrete and Computational Geometry, Chapter 65, pp. 1435–1463, 2nd edn. Chapman & Hall/CRC, Boca Raton (2004)
14. Mehlhorn, K., Näher, S.: LEDA: A Platform for Combinatorial and Geometric Computing. Cambridge University Press, Cambridge (2000)
15. Wein, R., Fogel, E., Zukerman, B., Halperin, D.: Advanced programming techniques applied to CGAL's arrangement package. Comput. Geom. Theor. Appl. **36**(1–2), 37–63 (2007)

Equivalence Between Priority Queues and Sorting

2002; Thorup

REZAUL A. CHOWDHURY
Department of Computer Sciences,
University of Texas at Austin,
Austin, TX, USA

Keywords and Synonyms

Heap

Problem Definition

A *priority queue* is an abstract data structure that maintains a set Q of elements, each with an associated value called a *key*, under the following set of operations [4,7].

insert(Q, x, k)**:** Inserts element x with key k into Q.
find-min(Q)**:** Returns an element of Q with the minimum key, but does not change Q.
delete(Q, x, k)**:** Deletes element x with key k from Q.

Additionally, the following operations are often supported.

delete-min(Q)**:** Deletes an element with the minimum key value from Q, and returns it.
decrease-key(Q, x, k)**:** Decreases the current key k' of x to k assuming $k < k'$.
meld(Q_1, Q_2)**:** Given priority queues Q_1 and Q_2, returns the priority queue $Q_1 \cup Q_2$.

Observe that a *delete-min* can be implemented as a *find-min* followed by a *delete*, a *decrease-key* as a *delete* followed by an *insert*, and a *meld* as a series of *find-min*, *delete* and *insert*. However, more efficient implementations of *decrease-key* and *meld* often exist [4,7].

Priority queues have many practical applications including event-driven simulation, job scheduling on a shared computer, and computation of shortest paths, minimum spanning forests, minimum cost matching, optimum branching, etc. [4,7].

A priority queue can trivially be used for sorting by first inserting all keys to be sorted into the priority queue and then by repeatedly extracting the current minimum. The major contribution in [15] is a reduction showing that the converse is also true. The results in [15] imply that priority queues are computationally equivalent to sorting, that is, asymptotically, the per key cost of sorting is the update time of a priority queue.

A result similar to those in [15] was presented in [14] which resulted in monotone priority queues (i. e., meaning that the extracted minimums are non-decreasing) with amortized time bounds only. In contrast, general priority queues with worst-case bounds are constructed in [15].

In addition to establishing the equivalence between priority queues and sorting, the reductions in [15] are also used to translate several known sorting results into new results on priority queues.

Background

Some relevant background information is summarized below which will be useful in understanding the key results in Sect. "Key Results".

- A standard *word RAM* models what one programs in a standard programming language such as C. In addition to direct and indirect addressing and conditional jumps, there are functions, such as addition and multiplication, operating on a constant number of words. The memory is divided into words, addressed linearly starting from 0. The running time of a program is the number of instructions executed and the space is the maximal address used. The word-length is a machine dependent parameter which is big enough to hold a key, and at least logarithmic in the number of input keys so that they can be addressed.
- A pointer machine is like the word RAM except that addresses cannot be manipulated.
- The AC^0 complexity class consists of constant-depth circuits with unlimited fan-in [18]. Standard AC^0 operations refer to the operations available via C, but where the functions on words are in AC^0. For example, this includes addition but not multiplication.
- Integer keys will refer to non-negative integers. However, if the input keys are signed integers, the correct ordering of the keys is obtained by flipping their sign bits and interpreting them as unsigned integers. Similar tricks work for floating point numbers and integer fractions [14].
- The atomic heaps of Fredman and Willard [6] are used in one of the reductions in [15]. These heaps can support updates and searches in sets of $O(\log^2 n)$ keys in $O(1)$ worst-case time [19]. However, atomic heaps use multiplication operations which are not in AC^0.

Key Results

The main results in this paper are two reductions from priority queues to sorting. The stronger of the two, stated in Theorem 1, is for integer priority queues running on a standard word RAM.

Theorem 1 *If for some non-decreasing function S, up to n integer keys can be sorted in S(n) time per key, an integer priority queue can be implemented supporting find-min in constant time, and updates, i. e., insert and delete, in $S(n) + O(1)$ time. Here n is the current number of keys in the queue. The reduction uses linear space. The reduction runs on a standard word RAM assuming that each integer key is contained in a single word.*

The reduction above provides the following new bounds for linear space integer priority queues improving previous bounds in [8,14] and [5], respectively.

1. **(Deterministic)** $O(\log\log n)$ update time using a sorting algorithm in [9].
2. **(Randomized)** $O\left(\sqrt{\log\log n}\right)$ expected update time using the sorting algorithm in [10].
3. **(Randomized with $O(1)$ *decrease-key*)** $O\left((\log n)^{\frac{1}{(2-\epsilon)}}\right)$ expected update time for word length $\geq \log n$ and any constant $\epsilon > 0$, using the sorting algorithm in [3].

The reduction in Theorem 1 employs atomic heaps [6] which in addition to being very complicated, use non-AC^0 operations. The following slightly weaker recursive reduction which does not restrict the domain of the keys is completely combinatorial.

Theorem 2 *Given a sorter that sorts up to n keys in S(n) time per key, a priority queue can be implemented supporting find-min in constant time, and updates in T(n) time where n is the current number of keys in the queue and T(n) satisfies the recurrence:*

$$T(n) = O\left(S(n) + T(\log^2 n)\right) .$$

The reduction runs on a pointer machine in linear space using only standard AC^0 operations. Key values are only accessed by the sorter.

This reduction implies the following new priority queue bounds not implied by Theorem 1, where the first two bounds improve previous bounds in [13] and [16], respectively.

1. **(Deterministic in Standard AC^0)** $O\left((\log\log n)^{1+\epsilon}\right)$ update time for any constant $\epsilon > 0$ using a standard AC^0 integer sorting algorithm in [10].
2. **(Randomized in Standard AC^0)** $O\left(\log\log n\right)$ expected update time using a standard AC^0 integer sorting algorithm in [16].
3. **(String of *l* Words)** $O(l + \log\log n)$ deterministic and $O\left(l + \sqrt{\log\log n}\right)$ randomized expected update time using the string sorting results in [10].

The Reduction in Theorem 1

Given a sorting routine that can sort up to n keys in $S(n)$ time per key, the priority queue is constructed as follows.

The data structure has two major components: a sorted list of keys and a set of update buffers. The key list is partitioned into small segments, each of which is maintained in an atomic heap allowing constant time update and search operations on that segment. Each update buffer has a different capacity and accumulates updates (*insert*/*delete*) with key values in a different range. Smaller update buffers accept updates with smaller keys. An atomic heap is used to determine in constant time which update buffer collects a new update. When an update buffer accumulates enough updates, they first enter a sorting phase and then a merging phase. In the merging phase each update is applied on the proper segment in the key list, and invariants on segment size and ranges of update buffers are fixed. These phases are not executed immediately, instead they are executed in fixed time increments over a period of time. An update buffer continues to accept new updates while some updates accepted by it earlier are still in the sorting phase, and some even older updates are in the merging phase. Every time it accepts a new update, $S(n)$ time is spent on the sorting phase associated with it, and $O(1)$ time on its merging phase. This strategy allows the sorting and merging phases to complete execution by the time the update buffer becomes full again, and thus keeping the movement of updates through different phases smooth while maintaining an $S(n) + O(1)$ worst-case time bound per update. Moreover, the size and capacity constraints ensure that the smallest key in the data structure is available in $O(1)$ time. More details are given below.

The Sorted Key List: The sorted key list contains most of the keys in the data structure including the minimum key, and is known as the *base list*. This list is partitioned into *base segments* containing $\Theta((\log n)^2)$ keys each. Keys in each segment are maintained in an atomic heap so that if a new update is known to apply to a particular segment it can be applied in $O(1)$ time. If a base segment becomes too large or too small, it is split or joined with an adjacent segment.

Update Buffers: The base segments are separated by *base splitters*, and $O(\log n)$ of them are chosen to become *top splitters* so that the number of keys in the base list below the ith ($i > 0$) top splitter s_i is $\Theta\left(4^i (\log n)^2\right)$. These splitters are placed in an atomic heap. As the base list changes the top splitters are moved, as needed, in order to maintain their exponential distribution.

Associated with each top splitter s_i, $i > 1$, are three buffers: an *entrance*, a *sorter*, and a *merger*, each with capacity for 4^i keys. Top splitter s_i works in a cycle of 4^i steps. The cycle starts with an empty entrance, at most 4^i updates in the sorter, and a sorted list of at most 4^i updates in the merger. In each step one may accept an update for the entrance, spend $O(4^i) = S(n)$ time in the sorter, and $O(1)$ time in merging the sorted list in the merger with the $O(4^i)$ base splitters in $[s_{i-2}, s_{i+1})$ (assuming $s_0 = 0$, $s_{-1} = -\infty$) and scanning for a new s_i among them. The implementation guarantees that all keys in the buffers of s_i lie in $[s_{i-2}, s_{i+1})$. Therefore, after 4^i such steps, the sorted list is correctly merged with the base list, a new s_i is found, and a new sorted list is produced. The sorter then takes the role of the merger, the entrance becomes the sorter, and the empty merger becomes the new entrance.

Handling Updates: When a new update key k (*insert*/*delete*) is received, the atomic heap of top splitters is used to find in $O(1)$ time the s_i such that $k \in [s_{i-1}, s_i)$. If $k \in [s_0, s_1)$, its position is identified among the $O(1)$ base splitters below s_1, and the corresponding base segment is updated in $O(1)$ time using the atomic heap over the keys of that segment. If $k \in [s_{i-1}, s_i)$ for some $i > 1$, the update is placed in the entrance of s_i, performing one step of the cycle of s_i in $S(n) + O(1)$ time. Additionally, during each update another splitter s_r is chosen in a round-robin fashion, and a fraction $1/\log n$ of a step of a cycle of s_r is executed in $O(1)$ time. The work on s_r ensures that after every $O((\log n)^2)$ updates some progress is made on moving each top splitter.

A *find-min* returns the minimum element of the base list which is available in $O(1)$ time.

The Reduction in Theorem 2

This reduction follows from the previous reduction by replacing all atomic heaps by the buffer systems developed for the top splitters.

Further Improvement

In [1] Alstrup et al. present a general reduction that transforms a priority queue to support *insert* in $O(1)$ time while keeping the other bounds unchanged. This reduction can be used to reduce the cost of insertion to a constant in Theorems 1 and 2.

Applications

The equivalence results in [15] can be used to translate known sorting results into new results on priority queues for integers and strings in different computational models (see Sect. "Key Results"). These results can also be viewed as a new means of proving lower bounds for sorting via priority queues.

A new RAM priority queue that matches the bounds in Theorem 1 and also supports *decrease-key* in $O(1)$ time is presented in [17]. This construction combines Andersson's exponential search trees [2] with the priority queues implied by Theorem 1. The reduction in Theorem 1 is also used in [12] in order to develop an adaptive integer sorting algorithm for the word RAM. Reductions from meldable priority queues to sorting presented in [11] use the reductions from non-meldable priority queues to sorting given in [15].

Open Problems

As noted before, the combinatorial reduction for pointer machines given in Theorem 2 is weaker than the word RAM reduction. For example, for a hypothetical linear time sorting algorithm, Theorem 1 implies a priority queue with an update time of $O(1)$ while Theorem 2 implies $2^{O(\log^* n)}$-time updates. Whether this gap can be reduced or removed is still an open question.

Cross References

► Cache-Oblivious Sorting
► External Sorting and Permuting
► String Sorting
► Suffix Tree Construction in RAM

Recommended Reading

1. Alstrup, S., Husfeldt, T., Rauhe, T., Thorup, M.: Black box for constant-time insertion in priority queues (note). ACM TALG **1**(1), 102–106 (2005)
2. Andersson, A.: Faster deterministic sorting and searching in linear space. In: Proc. 37th FOCS, 1998, pp. 135–141
3. Andersson, A., Hagerup, T., Nilsson, S., Raman, R.: Sorting in linear time? J. Comp. Syst. Sci. **57**, 74–93 (1998). Announced at STOC'95
4. Cormen, T., Leiserson, C., Rivest, R., Stein, C.: Introduction to Algorithms. MIT Press, Cambridge, MA (2001)
5. Fredman, M., Willard, D.: Surpassing the information theoretic bound with fusion trees. J. Comput. Syst. Sci. **47**, 424–436 (1993). Announced at STOC'90
6. Fredman, M., Willard, D.: Trans-dichotomous algorithms for minimum spanning trees and shortest paths. J. Comput. Syst. Sci. **48**, 533–551 (1994)

7. Fredman, M., Tarjan, R.: Fibonacci heaps and their uses in improved network optimization algorithms. J. ACM **34**(3), 596–615 (1987)
8. Han, Y.: Improved fast integer sorting in linear space. Inf. Comput. **170**(8), 81–94 (2001). Announced at STACS'00 and SODA'01
9. Han, Y.: Deterministic sorting in $O(n \log\log n)$ time and linear space. J. Algorithms **50**(1), 96–105 (2004). Announced at STOC'02
10. Han, Y., Thorup, M.: Integer sorting in $O(n\sqrt{\log\log n})$ expected time and linear space. In: Proc. 43rd FOCS, 2002, pp. 135–144
11. Mendelson, R., Tarjan, R., Thorup, M., Zwick, U.: Melding priority queues. ACM TALG **2**(4), 535–556 (2006). Announced at SODA'04
12. Pagh, A., Pagh, R., Thorup, M.: On adaptive integer sorting. In: Proc. 12th ESA, 2004, pp. 556–579
13. Thorup, M.: Faster deterministic sorting and priority queues in linear space. In: Proc. 9th SODA, 1998, pp. 550–555
14. Thorup, M.: On RAM priority queues. SIAM J. Comput. **30**(1), 86–109 (2000). Announced at SODA'96
15. Thorup, M.: Equivalence between priority queues and sorting. In: Proc. 43rd FOCS, 2002, pp. 125–134
16. Thorup, M.: Randomized sorting in $O(n \log\log n)$ time and linear space using addition, shift, and bit-wise boolean operations. J. Algorithms **42**(2), 205–230 (2002). Announced at SODA'97
17. Thorup, M.: Integer priority queues with decrease key in constant time and the single source shortest paths problem. J. Comput. Syst. Sci. (special issue on STOC'03) **69**(3), 330–353 (2004)
18. Vollmer, H.: Introduction to circuit complexity: a uniform approach. Springer, New York (1999)
19. Willard, D.: Examining computational geometry, van Emde Boas trees, and hashing from the perspective of the fusion tree. SIAM J. Comput. **29**(3), 1030–1049 (2000). Announced at SODA'92

Error-Control Codes, Reed–Muller Code

► Learning Heavy Fourier Coefficients of Boolean Functions

Error Correction

► Decoding Reed–Solomon Codes
► List Decoding near Capacity: Folded RS Codes
► LP Decoding

Euclidean Graphs and Trees

► Minimum Geometric Spanning Trees
► Minimum k-Connected Geometric Networks

Euclidean Traveling Salesperson Problem

1998; Arora

Artur Czumaj
DIMAP and Computer Science, University of Warwick, Coventry, UK

Keywords and Synonyms

Euclidean TSP; Geometric TSP; Geometric traveling salesman problem

Problem Definition

This entry considers geometric optimization $\mathcal{NP}$-hard problems like the Euclidean Traveling Salesperson problem and the Euclidean Steiner Tree problem. These problems are geometric variants of standard graph optimization problems, and the restriction of the input instances to geometric or Euclidean case arise in numerous applications (see [1,2]). The main focus of this chapter is the Euclidean Traveling Salesperson problem.

Notation

The *Euclidean Traveling Salesperson Problem (TSP)*: For a given set S of n points in the Euclidean space $\mathbb{R}^d$, find a path of minimum length that visits each point exactly once.

The cost $\delta(x, y)$ of an edge connecting a pair of points $x, y \in \mathbb{R}^d$ is equal to the Euclidean distance between points x and y. That is, $\delta(x, y) = \sqrt{\sum_{i=1}^{d}(x_i - y_i)^2}$, where $x = (x_1, \dots, x_d)$ and $y = (y_1, \dots, y_d)$. More generally, the distance can be defined using other norms, such as ℓ_p norms for any $p > 1$, $\delta(x, y) = \left(\sum_{i=1}^{d}(x_i - y_i)^p\right)^{1/p}$.

For a given set S of points in the Euclidean space $\mathbb{R}^d$, for an integer d, $d \geq 2$, an *Euclidean graph* (network) is a graph $G = (S, E)$, where E is the set of straight-line segments connecting pairs of points in S. If all pairs of points in S are connected by edges in E, then G is called a *complete Euclidean graph on* S. The cost of the graph is equal to the sum of the costs of the edges of the graph: $\text{cost}(G) = \sum_{(x,y)\in E} \delta(x, y)$.

A *polynomial-time approximation scheme (PTAS)* is a family of algorithms $\{\mathcal{A}_\varepsilon\}$ such that, for each fixed $\varepsilon > 0$, $\mathcal{A}_\varepsilon$ runs in a time which is a polynomial of the size of the input, and produces a $(1 + \varepsilon)$-approximation.

Related work

The classical book by Lawler et al. [12] provides extensive information about the TSP. Also, the survey exposition of Bern and Eppstein [7] presents state of the art research done on the geometric TSP up to 1995, and the survey of Arora [2] discusses the research done after 1995.

Key Results

In general graphs the TSP graph problem is well known to be $\mathcal{NP}$-hard, and the same claim holds for the Euclidean TSP problem [11], [14].

Theorem 1 *The Euclidean TSP problem is $\mathcal{NP}$-hard.*

Perhaps rather surprisingly, it is still not known if the decision version of the problem is $\mathcal{NP}$-complete [11]. (The decision version of the Euclidean TSP problem is for a given point set in the Euclidean space $\mathbb{R}^d$ and a number t, verify if there is a simple path of length smaller than t that visits each point exactly once.)

The approximability of TSP has been studied extensively over the last few decades. It is not hard to see that TSP is not approximable in polynomial-time (unless $\mathcal{P} = \mathcal{NP}$) for arbitrary graphs with arbitrary edge costs. When the weights satisfy the triangle inequality (the so called *metric TSP*), there is a polynomial-time 3/2-approximation algorithm due to Christofides [8], and it is known that no PTAS exists (unless $\mathcal{P} = \mathcal{NP}$). This result has been strengthened by Trevisan [17] to include Euclidean graphs in high-dimensions (the same result holds also for any ℓ_p metric).

Theorem 2 (Trevisan [17]) *If $d \geq \log n$, then there exists a constant $\epsilon > 0$ such that the Euclidean TSP problem in $\mathbb{R}^d$ is $\mathcal{NP}$-hard to approximate within a factor of $1 + \epsilon$.*

In particular, this result implies that if $d \geq \log n$, then the Euclidean TSP problem in $\mathbb{R}^d$ has no PTAS unless $\mathcal{P} = \mathcal{NP}$.

The same result also holds for any ℓ_p metric. Furthermore, Theorem 2 implies that the Euclidean TSP in $\mathbb{R}^{\log n}$ is APX PB-hard under E-reductions and APX-complete under AP-reductions.

It was believed that Theorem 2 might hold for smaller values of d, in particular even for $d = 2$, but this has been disproved independently by Arora [1] and Mitchell [13].

Theorem 3 (Arora [1], Mitchell [13]) *The Euclidean TSP on the plane has a PTAS.*

The main idea of the algorithms of Arora and Mitchell is rather simple, but the details of the analysis are quite complicated. Both algorithms follow the same approach. First, one proves a so-called *Structure Theorem*. This demonstrates that there is a $(1 + \epsilon)$-approximation that has some local properties. In the case of the Euclidean TSP problem, there is a quadtree partition of the space containing all the points, such that each cell of the quadtree is crossed by the tour at most a constant number of times, and only in some pre-specified locations. After proving the Structure Theorem, one uses dynamic programming to find an optimal (or almost optimal) solution that obeys the local properties specified in the Structure Theorem.

The original algorithms presented in the first conference version of [1] and in the early version of [13] have running times of the form $\mathcal{O}(n^{1/\epsilon})$ to obtain a $(1 + \epsilon)$-approximation, but this has been subsequently improved. In particular, Arora's randomized algorithm in [1] runs in time $\mathcal{O}(n(\log n)^{1/\epsilon})$, and it can be derandomized with a slow-down of $\mathcal{O}(n)$. The result from Theorem 3 can be also extended to higher dimensions. Arora shows the following result.

Theorem 4 (Arora [1]) *For every constant d, the Euclidean TSP in $\mathbb{R}^d$ has a PTAS.*

For every fixed $c > 1$ and given any n nodes in $\mathbb{R}^d$, there is a randomized algorithm that finds a $(1 + 1/c)$-approximation of the optimum traveling salesman tour in $\mathcal{O}(n\,(\log n)^{(\mathcal{O}(\sqrt{d}c))^{d-1}})$ time. In particular, for any constant d and c, the running time is $\mathcal{O}(n\,(\log n)^{\mathcal{O}(1)})$. The algorithm can be derandomized by increasing the running time by a factor of $\mathcal{O}(n^d)$.

This was later extended by Rao and Smith [15], who proved the following theorem.

Theorem 5 (Rao and Smith [15]) *There is a deterministic algorithm that computes a $(1 + 1/c)$-approximation of the optimum traveling salesman tour in $\mathcal{O}(2^{(cd)^{\mathcal{O}(d)}}\, n + (cd)^{\mathcal{O}(d)}\, n \log n)$ time.*

There is also a randomized Monte Carlo algorithm that succeeds with probability at least 1/2 and that computes a $(1 + 1/c)$-approximation of the optimum traveling salesman tour in the expected $(c\sqrt{d})^{\mathcal{O}(d(c\sqrt{d})^{d-1})}\, n + \mathcal{O}(d\, n \log n)$ time.

In the special and most interesting case, when $d = 2$, Rao and Smith show the following.

Theorem 6 (Rao and Smith [15]) *There is a deterministic algorithm that computes a $(1 + 1/c)$-approximation of the optimum traveling salesman tour in $\mathcal{O}(n\, 2^{c^{\mathcal{O}(1)}} + c^{\mathcal{O}(1)}\, n \log n)$ time.*

There is a randomized Monte Carlo algorithm (which fails with probability smaller than 1/2) that computes *a* $(1 + 1/c)$*-approximation of the optimum traveling salesman tour in the expected* $\mathcal{O}(n\, 2^{c^{\mathcal{O}(1)}} + n \log n)$ *time.*

Applications

The techniques developed by Arora [1] and Mitchell [13] found numerous applications in the design of polynomial-time approximation schemes for geometric optimization problems.

Euclidean Minimum Steiner Tree Problem For a given set S of n points in the Euclidean space $\mathbb{R}^d$, find the minimum cost network connecting all the points in S (where the cost of a network is equal to the sum of the lengths of the edges defining it).

Theorem 7 ([1], [15]) *For every constant d, the Euclidean Minimum Steiner tree problem in* $\mathbb{R}^d$ *has a PTAS.*

Euclidean *k*-median Problem For a given set S of n points in the Euclidean space $\mathbb{R}^d$ and an integer k, find k medians among the points in S so that the sum of the distances from each point in S to its closest median is minimized.

Theorem 8 ([5]) *For every constant d, the Euclidean k-median problem in* $\mathbb{R}^d$ *has a PTAS.*

Euclidean *k*-TSP Problem For a given set S of n points in the Euclidean space $\mathbb{R}^d$ and an integer k, find the shortest tour that visits at least k points in S.

Euclidean *k*-MST Problem For a given set S of n points in the Euclidean space $\mathbb{R}^d$ and an integer k, find the shortest tree that contains at least k points from S.

Theorem 9 ([1]) *For every constant d, the Euclidean k-TSP and the Euclidean k-MST problems in* $\mathbb{R}^d$ *have a PTAS.*

Euclidean Minimum-cost *k*-connected Subgraph Problem For a given set S of n points in the Euclidean space $\mathbb{R}^d$ and an integer k, find the minimum-cost subgraph (of the complete graph on S) that is k-connected

Theorem 10 ([9]) *For every constant d and constant k, the Euclidean minimum-cost k-connected subgraph problem in* $\mathbb{R}^d$ *has a PTAS.*

The technique developed by Arora [1] and Mitchell [13] also led to some quasi-polynomial-time approximation schemes, that is, the algorithms with the running time of $n^{\mathcal{O}(\log n)}$. For example, Arora and Karokostas [4] gave a quasi-polynomial-time approximation scheme for the Euclidean minimum latency problem, and Remy and Steger [16] gave a quasi-polynomial-time approximation scheme for the minimum-weight triangulation problem.

For more discussion, see the survey by Arora [2] and [10].

Extensions to Planar Graphs

The dynamic programming approach used by Arora [1] and Mitchell [13] is also related to the recent advances for a number of optimization problems for planar graphs. For example, Arora et al. [3] designed a PTAS for the TSP problem in weighted planar graphs, and there is a PTAS for the problem of finding a minimum-cost spanning 2-connected subgraph of a planar graph [6].

Open Problems

One interesting open problem is if the quasi-polynomial-time approximation schemes mentioned above (for the minimum latency and the minimum-weight triangulation problems) can be extended to obtain polynomial-time approximation schemes. For more open problems, see Arora [2].

Cross References

▶ Metric TSP
▶ Minimum *k*-Connected Geometric Networks
▶ Minimum Weight Triangulation

Recommended Reading

1. Arora, S.: Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. J. ACM **45**(5), 753–782 (1998)
2. Arora, S.: Approximation schemes for $\mathcal{NP}$-hard geometric optimization problems: A survey. Math. Program. Ser. B **97**, 43–69 (2003)
3. Arora, S., Grigni, M., Karger, D., Klein, P., Woloszyn, A..: A polynomial time approximation scheme for weighted planar graph TSP. In: Proc. 9th Annual ACM-SIAM Symposium on Discrete Algorithms, 1998, pp. 33–41
4. Arora, S., Karakostas, G.: Approximation schemes for minimum latency problems. In: Proc. 31st Annual ACM Symposium on Theory of Computing, 1999, pp. 688–693
5. Arora, S., Raghavan, P., Rao, S.: Approximation schemes for Euclidean *k*-medians and related problems. In: Proc. 30th Annual ACM Symposium on Theory of Computing, 1998, pp. 106–113
6. Berger, A., Czumaj, A., Grigni, M., Zhao, H.: Approximation schemes for minimum 2-connected spanning subgraphs in weighted planar graphs. In: Proc. 13th Annual European Symposium on Algorithms, 2005, pp. 472–483
7. Bern, M., Eppstein, D.: Approximation algorithms for geometric problems. In: Hochbaum, D. (ed.) Approximation Algorithms for NP-hard problems. PWS Publishing, Boston (1996)

8. Christofides, N.: Worst-case analysis of a new heuristic for the traveling salesman problem. In: Technical report, Graduate School of Industrial Administration. Carnegie-Mellon University, Pittsburgh (1976)
9. Czumaj, A., Lingas, A.: On approximability of the minimum-cost *k*-connected spanning subgraph problem. In: Proc. 10th Annual ACM-SIAM Symposium on Discrete Algorithms, 1999, pp. 281–290
10. Czumaj, A., Lingas, A.: Approximation schemes for minimum-cost *k*-connectivity problems in geometric graphs. In: Gonzalez, T.F. (eds.) Handbook of Approximation Algorithms and Metaheuristics. CRC Press, Boca Raton (2007)
11. Garey, M.R., Graham, R.L., Johnson, D.S.: Some NP-complete geometric problems. In: Proc. 8th Annual ACM Symposium on Theory of Computing, 1976, pp. 10–22
12. Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., Shmoys, D.B.: The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization. Wiley, Chichester (1985)
13. Mitchell, J.S.B.: Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric TSP, *k*-MST, and related problems. SIAM J. Comput. **28**(4), 1298–1309 (1999)
14. Papadimitriou, C.H.: Euclidean TSP is NP-complete. Theor. Comput. Sci. **4**, 237–244 (1977)
15. Rao, S.B., Smith, W.D.: Approximating geometrical graphs via "spanners" and "banyans". In: Proc. 30th Annual ACM Symposium on Theory of Computing, 1998, pp. 540–550
16. Remy, J., Steger, A.: A quasi-polynomial time approximation scheme for minimum weight triangulation. In: Proc. 38th Annual ACM Symposium on Theory of Computing, 2006
17. Trevisan, L.: When Hamming meets Euclid: the approximability of geometric TSP and Steiner Tree. SIAM J. Comput. **30**(2), 475–485 (2000)

Exact Algorithms for Dominating Set

2005; Fomin, Grandoni, Kratsch

DIETER KRATSCH
UFM MIM – LITA, Paul Verlaine University,
Metz, France

Keywords and Synonyms

Connected dominating set

Problem Definition

The dominating set problem is a classical NP-hard optimization problem which fits into the broader class of covering problems. Hundreds of papers have been written on this problem that has a natural motivation in facility location.

Definition 1 For a given undirected, simple graph $G = (V, E)$ a subset of vertices $D \subseteq V$ is called a *dominating set* if every vertex $u \in V - D$ has a neighbor in D. The minimum dominating set (MDS) problem is to find a *minimum dominating set* of G, i. e. a dominating set of G of minimum cardinality.

Problem 1 (MDS)
Input: *Undirected simple graph* $G = (V, E)$.
Output: *A minimum dominating set* D *of* G.

Various modifications of the dominating set problem are of interest, some of them obtained by putting additional constraints on the dominating set such as, for example, requesting it to be an independent set or to be connected. In graph theory there is a huge literature on domination dealing with the problem and its many modifications (see e. g. [9]). In graph algorithms the MDS problem and some of its modifications like Independent Dominating Set and Connected Dominating Set have been studied as benchmark problems for attacking NP-hard problems under various algorithmic approaches.

Known Results

The algorithmic complexity of MDS and its modifications when restricted to inputs from a particular graph class has been studied extensively (see e. g. [10]). Among others, it is known that MDS remains NP-hard on bipartite graphs, split graphs, planar graphs and graphs of maximum degree three. Polynomial time algorithms to compute a minimum dominating set are known, for example, for permutation, interval and k-polygon graphs. There is also a $O(4^k n^{O(1)})$ time algorithm to solve MDS on graphs of treewidth at most k.

The dominating set problem is one of the basic problems in parameterized complexity [3]; it is W[2]-complete and thus it is unlikely that the problem is fixed parameter tractable. On the other hand, the problem is fixed parameter tractable on planar graphs. Concerning approximation, MDS is equivalent to MINIMUM SET COVER under L-reductions. There is an approximation algorithm solving MDS within a factor of $1 + \ln |V|$ and it cannot be approximated within a factor of $(1 - \epsilon) \ln |V|$ for any $\epsilon > 0$, unless NP $\subset$ DTIME$(n^{\log \log n})$ [1].

Moderately Exponential Time Algorithms

If P $\neq$ NP then no polynomial time algorithm can solve MDS. Even worse, it has been observed in [7] that unless SNP$\subseteq$ SUBEXP (which is considered to be highly unlikely), there is not even a subexponential time algorithm solving the dominating set problem.

The trivial $O(2^n (n+m))$ algorithm, which simply checks all the 2^n vertex subsets as to whether they are dominating, clearly solves MDS. Three faster algorithms were established in 2004. The algorithm of Fomin et al. [7] uses a deep graph-theoretic result due to B. Reed, stating that every graph on n vertices with minimum degree at least three has a dominating set of size at most $3n/8$, to establish an $O(2^{0.955n})$ time algorithm solving MDS. The $O(2^{0.919n})$ time algorithm of Randerath and Schiermeyer [11] uses very nice ideas including matching techniques to restrict the search space. Finally, Grandoni [8] established an $O(2^{0.850n})$ time algorithm to solve MDS.

The work of Fomin, Grandoni, and Kratsch [5] presents a simple and easy to implement recursive branch & reduce algorithm to solve MDS. The running time of the algorithm is significantly faster than the ones stated for previous algorithms. This is heavily based on the analysis of the running time by measure & conquer, which is a method to analyze the worst case running time of (simple) branch & reduce algorithms based on a sophisticated choice of the measure of a problem instance.

Key Results

Theorem 1 *There is a branch & reduce algorithm solving* MDS *in time* $O(2^{0.610n})$ *using polynomial space.*

Theorem 2 *There is an algorithm solving* MDS *in time* $O(2^{0.598n}$ *using exponential space.*

The algorithms of Theorem 1 and 2 are simple consequences of a transformation from MDS to Minimum Set Cover (MSC) combined with new moderately exponential time algorithms for MSC.

Problem 2 (MSC)
Input: *Finite set* $\mathcal{U}$ *and a collection* $\mathcal{S}$ *of subsets* $S_1, S_2, \ldots S_t$ *of* $\mathcal{U}$.
Output: *A minimum set cover* $\mathcal{S}'$, *where* $\mathcal{S}' \subseteq \mathcal{S}$ *is a set cover of* $(\mathcal{U}, \mathcal{S})$ *if* $\bigcup_{S_i \in \mathcal{S}'} S_i = \mathcal{U}$.

Theorem 3 *There is a branch & reduce algorithm solving* MSC *in time* $O(2^{0.305(|\mathcal{U}|+|\mathcal{S}|)})$ *using polynomial space.*

Applying memorization to the polynomial space algorithm of Theorem 3 the running time can be improved as follows.

Theorem 4 *There is an algorithm solving* MSC *in time* $O(2^{0.299(|\mathcal{S}|+|\mathcal{U}|)})$ *using exponential space.*

The analysis of the worst case running time of the simple branch & reduce algorithm solving MSC (of Theorem 3) is done by a careful choice of the measure of a problem instance which allows one to obtain an upper bound that is significantly smaller than the one that could be obtained using the standard measure. The refined analysis leads to a collection of recurrences. Then random local search is used to compute the weights, used in the definition of the measure, aiming at the best achievable upper bound of the worst-case running time.

Since current tools to analyze the worst-case running time of branch & reduce algorithms do not seem to produce tight upper bounds, exponential lower bounds of the worst-case running time of the algorithm are of interest.

Theorem 5 *The worst-case running time of the branch & reduce algorithm solving* MDS *(see Theorem 1) is* $\Omega(2^{n/3})$.

Applications

There are various other NP-hard domination-type problems that can be solved by a moderately exponential time algorithm based on an algorithm solving Minimum Set Cover: any instance of the initial problem is transformed to an instance of MSC (preferably with $|\mathcal{U}| = |\mathcal{S}|$), and then the algorithm of Theorem 3 or 4 is used to solve MSC and thus the initial problem. Examples of such problems are Total Dominating Set, k-Dominating Set, k-Center and MDS on split graphs.

Measure & Conquer and the strongly related quasiconvex analysis of Eppstein [4] have been used to design and analyze a variety of moderately exponential time algorithms for NP-hard problems: optimization, counting and enumeration problems. See for example [2,6].

Open Problems

A number of problems related to the work of Fomin, Grandoni, and Kratsch remain open. Although for various graph classes there are algorithms to solve MDS which are faster than the one for general graphs (of Theorem 1 and 2), no such algorithm is known for solving MDS on bipartite graphs.

The worst-case running times of simple branch & reduce algorithms like those solving MDS and MSC remain unknown. In the case of the polynomial space algorithm solving MDS there is a large gap between the $O(2^{0.610n})$ upper bound and the $\Omega(2^{n/3})$ lower bound. The situation is similar for other branch & reduce algorithms. Consequently, there is a strong need for new and better tools to analyze the worst-case running time of branch & reduce algorithms.

Cross References

► Connected Dominating Set
► Data Reduction for Domination in Graphs
► Vertex Cover Search Trees

Recommended Reading

1. Ausiello, G., Crescenzi, P., Gambosi, G., Kann, V., Marchetti-Spaccalema, A., Protasi, M.: Complexity and Approximation. Springer, Berlin (1999)
2. Byskov, J.M.: Exact algorithms for graph colouring and exact satisfiability. Ph. D. thesis, University of Aarhus, Denmark (2004)
3. Downey, R.G., Fellows, M.R.: Parameterized complexity. Springer, New York (1999)
4. Eppstein, D.: Quasiconvex analysis of backtracking algorithms. In: Proceedings of SODA 2004, pp. 781–790
5. Fomin, F.V., Grandoni, F., Kratsch, D.: Measure and conquer: Domination – A case study. In: Proceedings of ICALP 2005. LNCS, vol. 3380, pp. 192–203. Springer, Berlin (2005)
6. Fomin, F.V., Grandoni, F., Kratsch, D.: Measure and Conquer: A simple $O(2^{0.288n})$ Independent Set Algorithm. In: Proceedings of SODA 2006, pp. 18–25
7. Fomin, F.V., Kratsch, D., Woeginger, G.J.: Exact (exponential) algorithms for the dominating set problem. In: Proceedings of WG 2004. LNCS, vol. 3353, pp. 245–256. Springer, Berlin (2004)
8. Grandoni, F.: Exact Algorithms for Hard Graph Problems. Ph. D. thesis, Università di Roma "Tor Vergata", Roma, Italy (2004)
9. Haynes, T.W., Hedetniemi, S.T., Slater, P.J.: Fundamentals of domination in graphs. Marcel Dekker, New York (1998)
10. Kratsch, D.: Algorithms. In: Haynes, T., Hedetniemi, S., Slater, P. (eds.) Domination in Graphs: Advanced Topics, pp. 191–231. Marcel Dekker, New York (1998)
11. Randerath, B., Schiermeyer, I.: Exact algorithms for MINIMUM DOMINATING SET. Technical Report, zaik-469, Zentrum für Angewandte Informatik Köln (2004)
12. Woeginger, G.J.: Exact algorithms for NP-hard problems: A survey. In: Combinatorial Optimization – Eureka, You Shrink. LNCS, vol. 2570, pp. 185–207. Springer, Berlin (2003)

Exact Algorithms for General CNF SAT
1998; Hirsch
2003; Schuler

Edward A. Hirsch
Laboratory of Mathematical Logic, Steklov Institute of Mathematics at St. Petersburg, St. Petersburg, Russia

Keywords and Synonyms

SAT; Boolean satisfiability

Problem Definition

The satisfiability problem (*SAT*) for Boolean formulas in conjunctive normal form (*CNF*) is one of the first **NP**-complete problems [2,13]. Since its **NP**-completeness currently leaves no hope for polynomial-time algorithms, the progress goes by decreasing the exponent. There are several versions of this parametrized problem that differ in the parameter used for the estimation of the running time.

Problem 1 (SAT) INPUT: *Formula F in CNF containing n variables, m clauses, and l literals in total.*

OUTPUT: *"Yes" if F has a* satisfying assignment, *i. e., a substitution of Boolean values for the variables that makes F true. "No" otherwise.*

The bounds on the running time of SAT algorithms can be thus given in the form $|F|^{O(1)} \cdot \alpha^n$, $|F|^{O(1)} \cdot \beta^m$, or $|F|^{O(1)} \cdot \gamma^l$, where $|F|$ is the length of a reasonable bit representation of F (i. e., the formal input to the algorithm). In fact, for the present algorithms the bases β and γ are constants while α is a function $\alpha(n, m)$ of the formula parameters (because no better constant than $\alpha = 2$ is known).

Notation

A formula in conjunctive normal form is a set of clauses (understood as the conjunction of these clauses), a clause is a set of literals (understood as the disjunction of these literals), and a literal is either a Boolean variable or the negation of a Boolean variable. A truth assignment assigns Boolean values (`false` or `true`) to one or more variables. An assignment is abbreviated as the list of literals that are made true under this assignment (for example, assigning `false` to x and `true` to y is denoted by $\neg x, y$). The result of the application of an assignment A to a formula F (denoted $F[A]$) is the formula obtained by removing the clauses containing the true literals from F and removing the falsified literals from the remaining clauses. For example, if $F = (x \vee \neg y \vee z) \wedge (y \vee \neg z)$, then $F[\neg x, y] = (z)$. A *satisfying* assignment for F is an assignment A such that $F[A] =$ `true`. If such an assignment exists, F is called *satisfiable*.

Key Results

Bounds for β and γ

General Approach and a Bound for β The trivial brute-force algorithm enumerating all possible assignments to the n variables runs in 2^n polynomial-time steps. Thus $\alpha \le 2$, and by trivial reasons also $\beta, \gamma \le 2$. In the early 1980s Monien and Speckenmeyer noticed that β could be made smaller[1]. Then Kullmann and Luckhardt [12] set up a framework for divide-and-conquer[2] algorithms for SAT that split the original problem into several (yet usu-

[1] They and other researchers also noticed that α could be made smaller for a special case of the problem where the length of each clause is bounded by a constant; the reader is referred to another entry (*Local search algorithms for k-SAT*) of the *Encyclopedia* for relevant references and algorithms.

[2] Also called *DPLL* due to the papers of Davis and Putnam [7] and Davis, Logemann, and Loveland [6].

ally a constant number of) subproblems by substituting the values of some variables and simplifying the obtained formulas. This line of research resulted in the following upper bounds for β and γ:

Theorem 1 (Hirsch [8]) *SAT can be solved in time*
1. $|F|^{O(1)} \cdot 2^{0.30897m}$;
2. $|F|^{O(1)} \cdot 2^{0.10299l}$.

A typical divide-and-conquer algorithm for SAT consists of two phases: *splitting* of the original problem into several subproblems (for example, reducing *SAT(F)* to *SAT(F[x])* and $SAT(F[\neg x])$) and *simplification* of the obtained subproblems using polynomial-time transformation rules that do not affect the satisfiability of the subproblems (i. e., they replace a formula by an *equi-satisfiable* one). The subproblems $F_1, \dots, F_k$ for splitting are chosen so that the corresponding recurrent inequality using the simplified problems $F'_1, \dots, F'_k$,

$$T(F) \le \sum_{i=1}^{k} T(F'_i) + \text{const} ,$$

gives a desired upper bound on the number of leaves in the recurrence tree and, hence, on the running time of the algorithm. In particular, in order to obtain the bound $|F|^{O(1)} \cdot 2^{0.30897m}$ one takes either two subproblems $F[x], F[\neg x]$ with recurrent inequality

$$t_m \le t_{m-3} + t_{m-4}$$

or four subproblems $F[x, y], F[x, \neg y], F[\neg x, y], F[\neg x, \neg y]$ with recurrent inequality

$$t_m \le 2t_{m-6} + 2t_{m-7}$$

where $t_i = \max_{m(G) \le i} T(G)$. The simplification rules used in the $|F|^{O(1)} \cdot 2^{0.30897m}$-time and the $|F|^{O(1)} \cdot 2^{0.10299l}$-time algorithms are as follows.

Simplification Rules

Elimination of 1-clauses If F contains a 1-clause (a), replace F by $F[a]$.

Subsumption If F contains two clauses C and D such that $C \subseteq D$, replace F by $F \setminus \{D\}$.

Resolution with Subsumption Suppose a literal a and clauses C and D are such that a is the only literal satisfying both conditions $a \in C$ and $\neg a \in D$. In this case, the clause $(C \cup D) \setminus \{a, \neg a\}$ is called the *resolvent by the literal a* of the clauses C and D and denoted by $R(C, D)$.

The rule is: if $R(C, D) \subseteq D$, replace F by $(F \setminus \{D\}) \cup \{R(C, D)\}$.

Elimination of a Variable by Resolution [7] Given a literal a, construct the formula $\mathrm{DP}_a(F)$ by
1. adding to F all resolvents by a;
2. removing from F all clauses containing a or $\neg a$.

The rule is: if $\mathrm{DP}_a(F)$ is not larger in m (resp., in l) than F, then replace F by $\mathrm{DP}_a(F)$.

Elimination of Blocked Clauses A clause C is *blocked* for a literal a w.r.t. F if C contains the literal a, and the literal $\neg a$ occurs only in the clauses of F that contain the negation of at least one of the literals occurring in $C \setminus \{a\}$. For a CNF-formula F and a literal a occurring in it, the assignment $I(a, F)$ is defined as

$$\{a\} \cup \{\text{literals } x \notin \{a, \neg a\} |\ \text{the clause } \{\neg a, x\} \text{ is blocked for } \neg a \text{ w.r.t. } F\} .$$

Lemma 2 (Kullmann [11])

(1) *If a clause C is blocked for a literal a w.r.t. F, then F and* $F \setminus \{C\}$ *are equi-satisfiable.*
(2) *Given a literal a, the formula F is satisfiable iff at least one of the formulas* $F[\neg a]$ *and* $F[I(a, F)]$ *is satisfiable.*

The first claim of the lemma is employed as a simplification rule.

Application of the Black and White Literals Principle Let P be a binary relation between literals and formulas in CNF such that for a variable v and a formula F, at most one of $P(v, F)$ and $P(\neg v, F)$ holds.

Lemma 3 *Suppose that each clause of F that contains a literal w satisfying* $P(w, F)$ *contains also at least one literal b satisfying* $P(\neg b, F)$. *Then F and* $F[\{l | P(\neg l, F)\}]$ *are equi-satisfiable.*

A Bound for γ To obtain the bound $|F|^{O(1)} \cdot 2^{0.10299l}$, it is enough to use a pair $F[\neg a], F[I(a, F)]$ of subproblems (see Lemma 2(2)) achieving the desired recurrent inequality $t_l \le t_{l-5} + t_{l-17}$ and to switch to the $|F|^{O(1)} \cdot 2^{0.30897m}$-time algorithm if there are none. A recent (much more technically involved) improvement to this algorithm [16] achieves the bound $|F|^{O(1)} \cdot 2^{0.0926l}$.

A Bound for α

Currently, no non-trivial constant upper bound for α is known. However, starting with [14] there was an interest to non-constant bounds. A series of randomized and deterministic algorithms showing successive improvements was developed, and at the moment the best possible bound is achieved by a deterministic divide-and-conquer algorithm employing the following recursive procedure. The idea behind it is a dichotomy: either each clause of the input formula can be shortened to its first k literals (then a k-CNF algorithm can be applied), or all these literals in one of the clauses can be assumed false. (This clause-shortening approach can be attributed to Schuler [15] who used it in a randomized fashion. The following version of the deterministic algorithm achieving the best known bound both for deterministic and randomized algorithms appears in [5].)

```
Procedure $S$
Input: a CNF formula $F$ and a positive integer $k$.
1. Assume $F$ consists of clauses $C_1, \dots, C_m$. Change each
   clause $C_i$ to a clause $D_i$ as follows: If $|C_i| > k$ then
   choose any $k$ literals in $C_i$ and drop the other literals;
   otherwise leave $C_i$ as is, i. e., $D_i = C_i$. Let $F'$ denote the
   resulting formula.
2. Test satisfiability of $F'$ using the $m \cdot \text{poly}(n) \cdot (2 - 2/(k+1))^n$-time
   $k$-CNF algorithm defined in [3].
3. If $F'$ is satisfiable, output "satisfiable" and halt. Otherwise,
   for each $i$, do the following:
   (a) Convert $F$ to $F_i$ as follows:
       i. Replace $C_j$ by $D_j$ for all $j < i$;
       ii. Assign false to all literals in $D_i$.
   (b) Recursively invoke Procedure $S$ on $(F_i, k)$.
4. Return "unsatisfiable".
```

The algorithm just invokes Procedure S on the original formula and the integer parameter $k = k*(m, n)$. The most accurate analysis of this family of algorithms by Calabro, Impagliazzo, and Paturi [1] implies that, assuming that $m > n$, one can obtain the following bound by taking $k(m, n) = 2\log(m/n) + \text{const}$. (This explicit bound is not stated in [1] and is inferred in [4].)

Theorem 4 (Dantsin, Hirsch [4]) *Assuming $m > n$, SAT can be solved in time*

$$|F|^{O(1)} \cdot 2^{n\left(1 - \frac{1}{O(\log(m/n))}\right)} .$$

Applications

While SAT has numerous applications, the presented algorithms have no direct effect on them.

Open Problems

Proving a constant upper bound on $\alpha < 2$ remains a major open problem in the field, as well as the hypothetic existence of $(1+\varepsilon)^l$-time algorithms for arbitrary small $\varepsilon > 0$.

It is possible to perform the analysis of a divide-and-conquer algorithm and even to generate simplification rules automatically [10]. However, this approach so far led to new bounds only for the (NP-complete) optimization version of 2-SAT [9].

Experimental Results

Jun Wang has implemented the algorithm yielding the bound on β and collected some statistics regarding the number of applications of the simplification rules [17].

Cross References

► Local Search Algorithms for kSAT
► Parameterized SAT

Recommended Reading

1. Calabro, C., Impagliazzo, R., Paturi, R.: A duality between clause width and clause density for SAT. In: Proceedings of the 21st Annual IEEE Conference on Computational Complexity (CCC 2006), pp. 252–260. IEEE Computer Society (2006)
2. Cook, S.A.: The Complexity of Theorem Proving Procedures. Proceedings of the Third Annual ACM Symposium on Theory of Computing, May 1971, pp. 151–158. ACM (2006)
3. Dantsin, E., Goerdt, A., Hirsch, E.A., Kannan, R., Kleinberg, J., Papadimitriou, C., Raghavan, P., Schöning, U.: A deterministic $(2-2/(k+1))^n$ algorithm for k-SAT based on local search. Theor. Comput. Sci. **289**(1), 69–83 (2002)
4. Dantsin, E., Hirsch, E.A.: Worst-Case Upper Bounds. In: Biere, A., van Maaren, H., Walsh, T. (eds.) Handbook of Satisfiability. IOS Press (2008) To appear
5. Dantsin, E., Hirsch, E.A., Wolpert, A.: Clause shortening combined with pruning yields a new upper bound for deterministic SAT algorithms. In: Proceedings of CIAC-2006. Lecture Notes in Computer Science, vol. 3998, pp. 60–68. Springer, Berlin (2006)
6. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem-proving. Commun. ACM **5**, 394–397 (1962)
7. Davis, M., Putnam, H.: A computing procedure for quantification theory. J. ACM **7**, 201–215 (1960)
8. Hirsch, E.A.: New worst-case upper bounds for SAT. J. Autom. Reason. **24**(4), 397–420 (2000)
9. Kojevnikov, A., Kulikov, A.: A New Approach to Proving Upper Bounds for MAX-2-SAT. Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2006), pp. 11–17. ACM, SIAM (2006)
10. Kulikov, A.: Automated Generation of Simplification Rules for SAT and MAXSAT. Proceedings of the Eighth International Conference on Theory and Applications of Satisfiability Testing (SAT 2005). Lecture Notes in Computer Science, vol. 3569, pp. 430–436. Springer, Berlin (2005)
11. Kullmann, O.: New methods for 3-{SAT} decision and worst-case analysis. Theor. Comput. Sci. **223**(1–2):1–72 (1999)
12. Kullmann, O., Luckhardt, H.: Algorithms for SAT/TAUT decision based on various measures, preprint, 71 pages, http://cs-svr1.swan.ac.uk/csoliver/papers.html (1998)
13. Levin, L.A.: Universal Search Problems. Проблемы передачи информации **9**(3), 265–266, (1973). In Russian. English translation in: Trakhtenbrot, B.A.: A Survey of Russian Approaches to

Perebor (Brute-force Search) Algorithms. Annals of the History of Computing **6**(4), 384–400 (1984)
14. Pudlák, P.: Satisfiability – algorithms and logic. In: Proceedings of the 23rd International Symposium on Mathematical Foundations of Computer Science, MFCS'98. Lecture Notes in Computer Science, vol. 1450, pp. 129–141. Springer, Berlin (1998)
15. Schuler, R.: An algorithm for the satisfiability problem of formulas in conjunctive normal form. J. Algorithms **54**(1), 40–44 (2005)
16. Wahlström, M.: An algorithm for the SAT problem for formulae of linear length. In: Proceedings of the 13th Annual European Symposium on Algorithms, ESA 2005. Lecture Notes in Computer Science, vol. 3669, pp. 107–118. Springer, Berlin (2005)
17. Wang, J.: Generating and solving 3-SAT, MSc Thesis. Rochester Institute of Technology, Rochester (2002)

Exact Graph Coloring Using Inclusion–Exclusion
2006; Björklund, Husfeldt

ANDREAS BJÖRKLUND, THORE HUSFELDT
Department of Computer Science, Lund University, Lund, Sweden

Keywords and Synonyms

Vertex coloring

Problem Definition

A *k-coloring* of a graph $G = (V, E)$ assigns one of k colors to each vertex such that neighboring vertices have different colors. This is sometimes called *vertex coloring*.

The smallest integer k for which the graph G admits a k-coloring is denoted $\chi(G)$ and called the *chromatic number*. The number of k-colorings of G is denoted $P(G;k)$ and called the *chromatic polynomial*.

Key Results

The central observation is that $\chi(G)$ and $P(G;k)$ can be expressed by an inclusion–exclusion formula whose terms are determined by the number of independent sets of induced subgraphs of G. For $X \subseteq V$, let $s(X)$ denote the number of nonempty independent vertex subsets disjoint from X, and let $s_r(X)$ denote the number of ways to choose r nonempty independent vertex subsets $S_1, \dots, S_r$ (possibly overlapping and with repetitions), all disjoint from X, such that $|S_1| + \dots + |S_r| = |V|$.

Theorem 1 *Let G be a graph on n vertices.*

1.
$$\chi(G) = \min_{k \in \{1,\dots,n\}} \left\{ k \colon \sum_{X \subseteq V} (-1)^{|X|} s(X)^k > 0 \right\} .$$

2. For $k = 1, \dots, k$,
$$P(G;k) = \sum_{r=1}^{k} \binom{k}{r} \Big(\sum_{X \subseteq V} (-1)^{|X|} s_r(X) \Big), \quad (k = 1, 2, \dots, n) .$$

The time needed to evaluate these expressions is dominated by the 2^n evaluations of $s(X)$ and $s_r(X)$, respectively. These values can be pre-computed in time and space within a polynomial factor of 2^n because they satisfy

$$s(X) = \begin{cases} 0, & \text{if } X = V , \\ s\big(X \cup \{v\}\big) + s\big(X \cup \{v\} \cup N(v)\big) + 1, & \text{for } v \notin X , \end{cases}$$

where $N(v)$ are the neighbors of v in G. Alternatively, the values can be computed using exponential-time, polynomial-space algorithms from the literature.

This leads to the following bounds:

Theorem 2 *For a graph G on n vertices, $\chi(G)$ and P(G;k) can be computed in*
1. time and space $2^n n^{O(1)}$.
2. time $O(2.2461^n)$ and polynomial space

An optimal coloring that achieves $\chi(G)$ can be found within the same bounds.

The techniques generalize to arbitrary families of subsets over a universe of size n, provided membership in the family can be decided in polynomial time.

Applications

In addition to being a fundamental problem in combinatorial optimization, graph coloring also arises in many applications, including register allocation and scheduling.

Cross References

Recommended Reading

1. Björklund, A., Husfeldt, T.: Exact algorithms for exact satisfiability and number of perfect matchings. In: Proc. 33rd ICALP. LNCS, vol. 4051, pp. 548–1559. Springer (2006). Algorithmica, doi:10.1007/s00453-007-9149-8
2. Björklund, A., Husfeldt, T., Koivisto, M.: Set partitioning via inclusion–exclusion. SIAM J. Comput.
3. Björklund, A., Husfeldt, T., Kaski, P., Koivisto, M.: Fourier meets Möbius: fast subset convolution. In: Proceedings of the 39th Annual ACM Symposium on Theory of Computing (STOC), San Diego, CA, June 11–13, 2007. Association for Computing Machinery, pp. 67–74. New York (2007)

Experimental Methods for Algorithm Analysis

2001; McGeoch

Catherine C. McGeoch
Department of Mathematics and Computer Science, Amherst College, Amherst, MA, USA

Keywords and Synonyms

Experimental algorithmics; Empirical algorithmics; Empirical analysis of algorithms; Algorithm engineering

Problem Definition

Experimental analysis of algorithms describes not a specific algorithmic problem, but rather an approach to algorithm design and analysis. It complements, and forms a bridge between, traditional *theoretical analysis*, and the application-driven methodology used in *empirical analysis*.

The traditional theoretical approach to algorithm analysis defines algorithm efficiency in terms of counts of dominant operations, under some abstract model of computation such as a RAM; the input model is typically either worst-case or average-case. Theoretical results are usually expressed in terms of asymptotic bounds on the function relating input size to number of dominant operations performed.

This contrasts with the tradition of empirical analysis that has developed primarily in fields such as operations research, scientific computing, and artificial intelligence. In this tradition, the efficiency of implemented programs is typically evaluated according to CPU or wall-clock times; inputs are drawn from real-world applications or collections of benchmark test sets, and experimental results are usually expressed in comparative terms using tables and charts.

Experimental analysis of algorithms spans these two approaches by combining the sensibilities of the theoretician with the tools of the empiricist. Algorithm and program performance can be measured experimentally according to a wide variety of *performance indicators*, including the dominant cost traditional to theory, bottleneck operations that tend to dominate running time, data structure updates, instruction counts, and memory access costs. A researcher in experimental analysis selects performance indicators most appropriate to the scale and scope of the specific research question at hand. (Of course time is not the only metric of interest in algorithm studies; this approach can be used to analyze other properties such as solution quality or space use.)

Input instances for experimental algorithm analysis may be randomly generated or derived from application instances. In either case, they typically are described in terms of a small- to medium-sized collection of *controlled parameters*. A primary goal of experimentation is to investigate the cause-and-effect relationship between input parameters and algorithm/program performance indicators.

Research goals of experimental algorithmics may include discovering functions (not necessarily asymptotic) that describe the relationship between input and performance, assessing the strengths and weaknesses of different algorithm/data structures/programming strategies, and finding best algorithmic strategies for different input categories. Results are typically presented and illustrated with graphs showing comparisons and trends discovered in the data.

The two terms "empirical" and "experimental", are often used interchangeably in the literature. Sometimes the terms "old style" and "new style" are used to describe, respectively, the empirical and experimental approaches to this type of research. The related term "algorithm engineering" refers to a systematic design process that takes an abstract algorithm all the way to an implemented program, with an emphasis on program efficiency. Experimental and empirical analysis is often used to guide the algorithm engineering process. The general term *algorithmics* can refer to both design and analysis in algorithm research.

Key Results

None

Applications

Experimental analysis of algorithms has been used to investigate research problems originating in theoretical computer science. One example arises in the average-case analysis of algorithms for the One-Dimensional Bin Packing problem. Experimental analyses have led to new theorems about the performance of the optimal algorithm; new asymptotic bounds on average-case performance of approximation algorithms; extensions of theoretical results to new models of inputs; and to new algorithms with tighter approximation guarantees. Another example is the experimental discovery of a type of phase-transition behavior for random instances of the 3CNF-Satisfiabilty problem, which has led to new ways to characterize the difficulty of problem instances.

A second application of experimental algorithmics is to find more realistic models of computation, and to design new algorithms that perform better on these models. One example is found in the development of new memory-based models of computation that give more accurate time predictions than traditional unit-cost models. Using these models, researchers have found new cache-efficient and I/O-efficient algorithms that exploit properties of the memory hierarchy to achieve significant reductions in running time.

Experimental analysis is also used to design and select algorithms that work best in practice, algorithms that work best on specific categories of inputs, and algorithms that are most robust with respect to bad inputs.

Data Sets

Many repositories for data sets and instance generators to support experimental research are available on the Internet. They are usually organized according to specific combinatorial problems or classes of problems.

URL to Code

Many code repositories to support experimental research are available on the Internet. They are usually organized according to specific combinatorial problems or classes of problems. Skiena's *Stony Brook Algorithm Repository* (www.cs.sunysb.edu/~algorith/) provides a comprehensive collection of problem definitions and algorithm descriptions, with numerous links to implemented algorithms.

Recommended Reading

The algorithmic literature containing examples of experimental research is much too large to list here. Some articles containing advice and commentary on experimental methodology in the context of algorithm research appear in the list below.

The workshops and journals listed below are specifically intended to support research in experimental analysis of algorithms. Experimental work also appears in more general algorithm research venues such as SODA (ACM/IEEE Symposium on Data Structures and Algorithms), *Algorithmica*, and *ACM Transactions on Algorithms*.

1. *ACM Journal of Experimental Algorithmics*. Launched in 1996, this journal publishes contributed articles as well as special sections containing selected papers from ALENEX and WEA. Visit www.jea.acm.org, or visit portal.acm.org and click on ACM Digital Library/Journals/Journal of Experimental Algorithmics
2. ALENEX. Beginning in 1999, the annual workshop on Algorithm Engineering and Experimentation is sponsored by SIAM and ACM. It is co-located with SODA, the SIAM Symposium on Data Structures and Algorithms. Workshop proceedings are published in the Springer LNCS series. Visit www.siam.org/meetings/ for more information
3. Barr, R.S., Golden, B.L., Kelly, J.P., Resende, M.G.C., Stewart, W.R.: Designing and reporting on computational experiments with heuristic methods. J. Heuristic **1**(1), 9–32 (1995)
4. Cohen, P.R.: Empirical Methods for Artificial Intelligence. MIT Press, Cambridge (1995)
5. DIMACS Implementation Challenges. Each DIMACS Implementation Challenge is a year-long cooperative research event in which researchers cooperate to find the most efficient algorithms and strategies for selected algorithmic problems. The DIMACS Challenges since 1991 have targeted a variety of optimization problems on graphs; advanced data structures; and scientific application areas involving computational biology and parallel computation. The DIMACS Challenge proceedings are published by AMS as part of the DIMACS Series in Discrete Mathematics and Theoretical Computer Science. Visit dimacs.rutgers.edu/Challenges for more information
6. Johnson, D.S.: A theoretician's guide to the experimental analysis of algorithms. In: Goodrich, M.H., Johnson, D.S., McGeoch, C.C. (eds.) Data Structures, Near Neighbors Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 59. American Mathematical Society, Providence (2002)
7. McGeoch, C.C.: Toward an experimental method for algorithm simulation. INFORMS J. Comp. **1**(1), 1–15 (1996)
8. WEA. Beginning in 2001, the annual Workshop on Experimental and Efficient Algorithms is sponsored by EATCS. Workshop proceedings are published in the Springer LNCS series

External Memory

► I/O-model
► R-Trees

External Sorting and Permuting

1988; Aggarwal, Vitter

JEFFREY SCOTT VITTER
Department of Computer Science, Purdue University, West Lafayette, IN, USA

Keywords and Synonyms

Out-of-core sorting

Problem Definition

Notations The main properties of magnetic disks and multiple disk systems can be captured by the commonly used *parallel disk model* (PDM), which is summarized below in its current form as developed by Vitter and Shriver [16]:

$$\begin{aligned} N &= \text{problem size (in units of data items)}\,; \\ M &= \text{internal memory size (in units of data items)}\,; \\ B &= \text{block transfer size (in units of data items)}\,; \\ D &= \text{number of independent disk drives}\,; \\ P &= \text{number of CPUs}\,, \end{aligned}$$

where $M < N$, and $1 \le DB \le M/2$. The data items are assumed to be of fixed length. In a single I/O, each of the D disks can simultaneously transfer a block of B contiguous data items. (In the original 1988 article [2], the D blocks per I/O were allowed to come from the same disk, which is not realistic.) If $P \le D$, each of the P processors can drive about D/P disks; if $D < P$, each disk is shared by about P/D processors. The internal memory size is M/P per processor, and the P processors are connected by an interconnection network.

It is convenient to refer to some of the above PDM parameters in units of disk blocks rather than in units of data items; the resulting formulas are often simplified. We define the lowercase notation

$$n = \frac{N}{B}, \qquad m = \frac{M}{B}, \qquad q = \frac{Q}{B}, \qquad z = \frac{Z}{B} \tag{1}$$

to be the problem input size, internal memory size, query specification size, and query output size, respectively, in units of disk blocks.

The primary measures of performance in PDM are
1. the number of I/O operations performed,
2. the amount of disk space used, and
3. the internal (sequential or parallel) computation time.

For reasons of brevity in this survey, focus is restricted to only the first two measures. Most of the algorithms run in optimal CPU time, at least for the single-processor case. Ideally algorithms and data structures should use linear space, which means $O(N/B) = O(n)$ disk blocks of storage.

Problem 1 (External sorting) INPUT: *The input data records R_0, R_1, R_2, ... are initially "striped" across the D disks, in units of blocks, so that record R_i is in block $\lfloor i/B \rfloor$, and block j is stored on disk $j \bmod D$.*

OUTPUT: *A striped representation of a permuted ordering $R_{\sigma(0)}$, $R_{\sigma(1)}$, $R_{\sigma(2)}$, ... of the input records with the property that* $\mathrm{key}(R_{\sigma(i)}) \le \mathrm{key}(R_{\sigma(i+1)})$ *for all $i \ge 0$.*

Permuting is the special case of sorting in which the permutation that describes the final position of the records is given explicitly and does not have to be discovered, for example, by comparing keys.

Problem 2 (Permuting) INPUT: *Same input assumptions as in external sorting. In addition, a permutation σ of the integers $\{0, 1, 2, \ldots, N-1\}$ is specified.*

OUTPUT: *A striped representation of a permuted ordering $R_{\sigma(0)}, R_{\sigma(1)}, R_{\sigma(2)}, \ldots$ of the input records.*

Key Results

Theorem 1 ([2,12]) *The average-case and worst-case number of I/Os required for sorting $N = nB$ data items using D disks is*

$$\mathrm{Sort}(N) = \Theta\left(\frac{n}{D}\log_m n\right). \tag{2}$$

Theorem 2 ([2]) *The average-case and worst-case number of I/Os required for permuting N data items using D disks is*

$$\Theta\left(\min\left\{\frac{N}{D}, \mathrm{Sort}(N)\right\}\right). \tag{3}$$

Matrix transposition is the special case of permuting in which the permutation can be represented as a transposition of a matrix from row-major order into column-major order.

Theorem 3 ([2]) *With D disks, the number of I/Os required to transpose a $p \times q$ matrix from row-major order to column-major order is*

$$\Theta\left(\frac{n}{D}\log_m \min\{M, p, q, n\}\right), \tag{4}$$

where $N = pq$ and $n = N/B$.

Matrix transposition is a special case of a more general class of permutations called *bit-permute/complement* (BPC) permutations, which in turn is a subset of the class of *bit-matrix-multiply/complement* (BMMC) permutations. BMMC permutations are defined by a $\log N \times \log N$ nonsingular 0-1 matrix A and a $(\log N)$-length 0-1 vector c. An item with binary address x is mapped by the permutation to the binary address given by $Ax \oplus c$, where $\oplus$ denotes bitwise exclusive-or. BPC permutations are the special case of BMMC permutations in which A is a permutation matrix, that is, each row and each column of A contain a single 1. BPC permutations include matrix transposition, bit-reversal permutations (which arise in the FFT), vector-reversal permutations, hypercube permutations, and matrix reblocking. Cormen et al. [6] characterize the optimal number of I/Os needed to perform any given BMMC permutation solely as a function of the associated matrix A, and they give an optimal algorithm for implementing it.

Theorem 4 ([6]) *With D disks, the number of I/Os required to perform the BMMC permutation defined by matrix A and vector c is*

$$\Theta\left(\frac{n}{D}\left(1+\frac{\operatorname{rank}(\gamma)}{\log m}\right)\right), \tag{5}$$

where γ is the lower-left $\log n \times \log B$ submatrix of A.

The two main paradigms for external sorting are *distribution* and *merging*, which are discussed in the following sections for the PDM model.

Sorting by Distribution

Distribution sort [9] is a recursive process that uses a set of $S-1$ partitioning elements to partition the items into S disjoint buckets. All the items in one bucket precede all the items in the next bucket. The sort is completed by recursively sorting the individual buckets and concatenating them together to form a single fully sorted list.

One requirement is to choose the $S-1$ partitioning elements so that the buckets are of roughly equal size. When that is the case, the bucket sizes decrease from one level of recursion to the next by a relative factor of $\Theta(S)$, and thus there are $O(\log_S n)$ levels of recursion. During each level of recursion, the data are scanned. As the items stream through internal memory, they are partitioned into S buckets in an online manner. When a buffer of size B fills for one of the buckets, its block is written to the disks in the next I/O, and another buffer is used to store the next set of incoming items for the bucket. Therefore, the maximum number of buckets (and partitioning elements) is $S=\Theta(M/B)=\Theta(m)$, and the resulting number of levels of recursion is $\Theta(\log_m n)$. How to perform each level of recursion in a linear number of I/Os is discussed in [2,11,16].

An even better way to do distribution sort, and deterministically at that, is the BalanceSort method developed by Nodine and Vitter [11]. During the partitioning process, the algorithm keeps track of how evenly each bucket has been distributed so far among the disks. It maintains an invariant that guarantees good distribution across the disks for each bucket.

The distribution sort methods mentioned above for parallel disks perform write operations in complete stripes, which make it easy to write parity information for use in error correction and recovery. But since the blocks written in each stripe typically belong to multiple buckets, the buckets themselves will not be striped on the disks, and thus the disks must be used independently during read operations. In the write phase, each bucket must therefore keep track of the last block written to each disk so that the blocks for the bucket can be linked together.

An orthogonal approach is to stripe the contents of each bucket across the disks so that read operations can be done in a striped manner. As a result, the write operations must use disks independently, since during each write, multiple buckets will be writing to multiple stripes. Error correction and recovery can still be handled efficiently by devoting to each bucket one block-sized buffer in internal memory. The buffer is continuously updated to contain the exclusive-or (parity) of the blocks written to the current stripe, and after $D-1$ blocks have been written, the parity information in the buffer can be written to the final (Dth) block in the stripe.

Under this new scenario, the basic loop of the distribution sort algorithm is, as before, to read one memoryload at a time and partition the items into S buckets. However, unlike before, the blocks for each individual bucket will reside on the disks in contiguous stripes. Each block therefore has a predefined place where it must be written. With the normal round-robin ordering for the stripes (namely, $\ldots, 1, 2, 3, \ldots, D, 1, 2, 3, \ldots, D, \ldots$), the blocks of different buckets may "collide," meaning that they need to be written to the same disk, and subsequent blocks in those same buckets will also tend to collide. Vitter and Hutchinson [15] solve this problem by the technique of *randomized cycling*. For each of the S buckets, they determine the ordering of the disks in the stripe for that bucket via a random permutation of $\{1, 2, \ldots, D\}$. The S random permutations are chosen independently. If two blocks (from different buckets) happen to collide during a write to the same disk, one block is written to the disk and the other is kept on a write queue. With high probability, subsequent blocks in those two buckets will be written to different disks and thus will not collide. As long as there is a small pool of available buffer space to temporarily cache the blocks in the write queues, Vitter and Hutchinson [15] show that with high probability the writing proceeds optimally.

The randomized cycling method or the related merge sort methods discussed at the end of Section Sorting by Merging are the methods of choice for sorting with parallel disks. Distribution sort algorithms may have an advantage over the merge approaches presented in Section Sorting by Merging in that they typically make better use of lower levels of cache in the memory hierarchy of real systems, based upon analysis of distribution sort and merge sort algorithms on models of hierarchical memory.

Sorting by Merging

The *merge* paradigm is somewhat orthogonal to the distribution paradigm of the previous section. A typical merge sort algorithm works as follows [9]: In the "run formation" phase, the n blocks of data are scanned, one memoryload at a time; each memoryload is sorted into a single "run," which is then output onto a series of stripes on the disks. At the end of the run formation phase, there are $N/M = n/m$ (sorted) runs, each striped across the disks. (In actual implementations, "replacement-selection" can be used to get runs of $2M$ data items, on the average, when $M \gg B$ [9].) After the initial runs are formed, the merging phase begins. In each pass of the merging phase, R runs are merged at a time. For each merge, the R runs are scanned and its items merged in an online manner as they stream through internal memory. Double buffering is used to overlap I/O and computation. At most $R = \Theta(m)$ runs can be merged at a time, and the resulting number of passes is $O(\log_m n)$.

To achieve the optimal sorting bound (2), each merging pass must be done in $O(n/D)$ I/Os, which is easy to do for the single-disk case. In the more general multiple-disk case, each parallel read operation during the merging must on the average bring in the next $\Theta(D)$ blocks needed for the merging. The challenge is to ensure that those blocks reside on different disks so that they can be read in a single I/O (or a small constant number of I/Os). The difficulty lies in the fact that the runs being merged were themselves formed during the previous merge pass. Their blocks were written to the disks in the previous pass without knowledge of how they would interact with other runs in later merges.

The Greed Sort method of Nodine and Vitter [12] was the first optimal deterministic EM algorithm for sorting with multiple disks. It works by relaxing the merging process with a final pass to fix the merging. Aggarwal and Plaxton [1] developed an optimal deterministic merge sort based upon the Sharesort hypercube parallel sorting algorithm. To guarantee even distribution during the merging, it employs two high-level merging schemes in which the scheduling is almost oblivious. Like Greed Sort, the Sharesort algorithm is theoretically optimal (i. e., within a constant factor of optimal), but the constant factor is larger than the distribution sort methods.

One of the most practical methods for sorting is based upon the *simple randomized merge sort* (SRM) algorithm of Barve et al. [5], referred to as "randomized striping" by Knuth [9]. Each run is striped across the disks, but with a random starting point (the only place in the algorithm where randomness is utilized). During the merging process, the next block needed from each disk is read into memory, and if there is not enough room, the least needed blocks are "flushed" (without any I/Os required) to free up space.

Further improvements in merge sort are possible by a more careful prefetching schedule for the runs. Barve et al. [4], Kallahalla and Varman [8], Shah et al. [13], and others have developed competitive and optimal methods for prefetching blocks in parallel I/O systems. Hutchinson et al. [7] have demonstrated a powerful duality between parallel writing and parallel prefetching, which gives an easy way to compute optimal prefetching and caching schedules for multiple disks. More significantly, they show that the same duality exists between distribution and merging, which they exploit to get a provably optimal and very practical parallel disk merge sort. Rather than use random starting points and round-robin stripes as in SRM, Hutchinson et al. [7] order the stripes for each run independently, based upon the randomized cycling strategy discussed in Section Sorting by Distribution for distribution sort.

Handling Duplicates: Bundle Sorting

For the problem of *duplicate removal*, in which there are a total of K distinct items among the N items, Arge et al. [3] use a modification of merge sort to solve the problem in $O\big(n \max\{1, \log_m (K/B)\}\big)$ I/Os, which is optimal in the comparison model. When duplicates get grouped together during a merge, they are replaced by a single copy of the item and a count of the occurrences. The algorithm can be used to sort the file, assuming that a group of equal items can be represented by a single item and a count.

A harder instance of sorting called *bundle sorting* arises when there are K distinct key values among the N items, but all the items have different secondary information that must be maintained, and therefore items cannot be aggregated with a count. Matias et al. [10] develop optimal distribution sort algorithms for bundle sorting using

$$O\big(n \max\{1, \log_m \min\{K, n\}\}\big) \tag{6}$$

I/Os and prove the matching lower bound. They also show how to do bundle sorting (and sorting in general) *in place* (i. e., without extra disk space).

Permuting and Transposition

Permuting is the special case of sorting in which the key values of the N data items form a permutation of $\{1, 2, \ldots, N\}$. The I/O bound (3) for permuting can be realized by one of the optimal sorting algorithms except in the extreme case $B \log m = o(\log n)$, where it is faster to move the data items one by one in a nonblocked way. The one-by-one method is trivial if $D = 1$, but with multiple disks there may be bottlenecks on individual disks; one solution for doing the permuting in $O(N/D)$ I/Os is to apply the randomized balancing strategies of [16].

Matrix transposition can be as hard as general permuting when B is relatively large (say, $1/2M$) and N is $O(M^2)$, but for smaller B, the special structure of the transposition permutation makes transposition easier. In particular, the matrix can be broken up into square submatrices of B^2 elements such that each submatrix contains B blocks of the matrix in row-major order and also B blocks of the matrix in column-major order. Thus, if $B^2 < M$, the transpositions can be done in a simple one-pass operation by transposing the submatrices one at a time in internal memory.

Fast Fourier Transform and Permutation Networks

Computing the fast Fourier transform (FFT) in external memory consists of a series of I/Os that permit each computation implied by the FFT directed graph (or butterfly) to be done while its arguments are in internal memory. A permutation network computation consists of an oblivious (fixed) pattern of I/Os such that any of the $N!$ possible permutations can be realized; data items can only be reordered when they are in internal memory. A permutation network can be realized by a series of three FFTs.

The algorithms for FFT are faster and simpler than for sorting because the computation is nonadaptive in nature, and thus the communication pattern is fixed in advance [16].

Lower Bounds on I/O

The following proof of the permutation lower bound (3) of Theorem 2 is due to Aggarwal and Vitter [2]. The idea of the proof is to calculate, for each $t \geq 0$, the number of distinct orderings that are realizable by sequences of t I/Os. The value of t for which the number of distinct orderings first exceeds $N!/2$ is a lower bound on the average number of I/Os (and hence the worst-case number of I/Os) needed for permuting.

Assuming for the moment that there is only one disk, $D = 1$, consider how the number of realizable orderings can change as a result of an I/O. In terms of increasing the number of realizable orderings, the effect of reading a disk block is considerably more than that of writing a disk block, so it suffices to consider only the effect of read operations. During a read operation, there are at most B data items in the read block, and they can be interspersed among the M items in internal memory in at most $\binom{M}{B}$ ways, so the number of realizable orderings increases by a factor of $\binom{M}{B}$. If the block has never before resided in internal memory, the number of realizable orderings increases by an extra $B!$ factor, since the items in the block can be permuted among themselves. (This extra contribution of $B!$ can only happen once for each of the N/B original blocks.) There are at most $n + t \leq N \log N$ ways to choose which disk block is involved in the tth I/O (allowing an arbitrary amount of disk space). Hence, the number of distinct orderings that can be realized by all possible sequences of t I/Os is at most

$$(B!)^{N/B} \left(N(\log N) \binom{M}{B} \right)^t . \tag{7}$$

Setting the expression in (7) to be at least $N!/2$, and simplifying by taking the logarithm, the result is

$$N \log B + t \left(\log N + B \log \frac{M}{B} \right) = \Omega(N \log N) . \tag{8}$$

Solving for t gives the matching lower bound $\Omega(n \log_m n)$ for permuting for the case $D = 1$. The general lower bound (3) of Theorem 2 follows by dividing by D.

A stronger lower bound follows from a more refined argument that counts input operations separately from output operations [7]. For the typical case in which $B \log m = \omega(\log N)$, the I/O lower bound, up to lower order terms, is $2n \log_m n$. For the pathological in which $B \log m = o(\log N)$, the I/O lower bound, up to lower order terms, is N/D.

Permuting is a special case of sorting, and hence, the permuting lower bound applies also to sorting. In the unlikely case that $B \log m = o(\log n)$, the permuting bound is only $\Omega(N/D)$, and in that case the comparison model must be used to get the full lower bound (2) of Theorem 1 [2]. In the typical case in which $B \log m = \Omega(\log n)$, the comparison model is not needed to prove the sorting lower bound; the difficulty of sorting in that case arises not from determining the order of the data but from permuting (or routing) the data.

The proof used above for permuting also works for permutation networks, in which the communication pattern is oblivious (fixed). Since the choice of disk block is fixed for each t, there is no $N \log N$ term as there is in (7), and correspondingly there is no additive $\log N$ term in the inner expression as there is in (8). Hence, solving for t gives the lower bound (2) rather than (3). The lower bound follows directly from the counting argument; unlike the sorting derivation, it does not require the comparison model for the case $B \log m = o(\log n)$. The lower bound also applies directly to FFT, since permutation networks can be formed from three FFTs in sequence. The transposition lower bound involves a potential argument based upon a togetherness relation [2].

For the problem of bundle sorting, in which the N items have a total of K distinct key values (but the secondary information of each item is different), Matias et al. [10] derive the matching lower bound.

The lower bounds mentioned above assume that the data items are in some sense "indivisible," in that they are not split up and reassembled in some magic way to get the desired output. It is conjectured that the sorting lower bound (2) remains valid even if the indivisibility assumption is lifted. However, for an artificial problem related to transposition, removing the indivisibility assumption can lead to faster algorithms. Whether the conjecture is true is a challenging theoretical open problem.

Applications

Sorting and sorting-like operations account for a significant percentage of computer use [9], with numerous database applications. In addition, sorting is an important paradigm in the design of efficient EM algorithms, as shown in [14], where several applications can be found. With some technical qualifications, many problems that can be solved easily in linear time in internal memory, such as permuting, list ranking, expression tree evaluation, and finding connected components in a sparse graph, require the same number of I/Os in PDM as does sorting.

Open Problems

Several interesting challenges remain. One difficult theoretical problem is to prove lower bounds for permuting and sorting without the indivisibility assumption. Another question is to determine the I/O cost for each individual permutation, as a function of some simple characterization of the permutation, such as number of inversions. A continuing goal is to develop optimal EM algorithms and to translate theoretical gains into observable improvements in practice. Many interesting challenges and opportunities in algorithm design and analysis arise from new architectures being developed, such as networks of workstations, hierarchical storage devices, disk drives with processing capabilities, and storage devices based upon microelectromechanical systems (MEMS). Active (or intelligent) disks, in which disk drives have some processing capability and can filter information sent to the host, have recently been proposed to further reduce the I/O bottleneck, especially in large database applications. MEMS-based nonvolatile storage has the potential to serve as an intermediate level in the memory hierarchy between DRAM and disks. It could ultimately provide better latency and bandwidth than disks, at less cost per bit than DRAM.

URL to Code

Two systems for developing external memory algorithms are TPIE and STXXL, which can be downloaded from http://www.cs.duke.edu/TPIE/ and http://sttxl.sourceforge.net/, respectively. Both systems include subroutines for sorting and permuting and facilitate development of more advanced algorithms.

Cross References

► I/O-model

Recommended Reading

1. Aggarwal, A., Plaxton, C.G.: Optimal parallel sorting in multi-level storage. In: Proceedings of the ACM-SIAM Symposium on Discrete Algorithms, vol. 5, pp. 659–668. ACM Press, New York (1994)
2. Aggarwal, A., Vitter, J.S.: The Input/Output complexity of sorting and related problems. In: Communications of the ACM, 31 (1988), pp. 1116–1127. ACM Press, New York (1988)
3. Arge, L., Knudsen, M., Larsen, K.: A general lower bound on the I/O-complexity of comparison-based algorithms. In: Proceedings of the Workshop on Algorithms and Data Structures. Lect. Notes Comput. Sci. **709**, 83–94 (1993)
4. Barve, R.D., Kallahalla, M., Varman, P.J., Vitter, J.S.: Competitive analysis of buffer management algorithms. J. Algorithms **36**, 152–181 (2000)
5. Barve, R.D., Vitter, J.S.: A simple and efficient parallel disk mergesort. ACM Trans. Comput. Syst. **35**, 189–215 (2002)
6. Cormen, T.H., Sundquist, T., Wisniewski, L.F.: Asymptotically tight bounds for performing BMMC permutations on parallel disk systems. SIAM J. Comput. **28**, 105–136 (1999)
7. Hutchinson, D.A., Sanders, P., Vitter, J.S.: Duality between prefetching and queued writing with parallel disks. SIAM J. Comput. **34**, 1443–1463 (2005)
8. Kallahalla, M., Varman, P.J.: Optimal read-once parallel disk scheduling. Algorithmica **43**, 309–343 (2005)
9. Knuth, D.E.: Sorting and Searching. The Art of Computer Programming, vol. 3, 2nd edn. Addison-Wesley, Reading (1998)
10. Matias, Y., Segal, E., Vitter, J.S.: Efficient bundle sorting. SIAM J. Comput. **36**(2), 394–410 (2006)
11. Nodine, M.H., Vitter, J.S.: Deterministic distribution sort in shared and distributed memory multiprocessors. In: Proceedings of the ACM Symposium on Parallel Algorithms and Architectures, June–July 1993, vol. 5, pp. 120–129, ACM Press, New York (1993)
12. Nodine, M.H., Vitter, J.S.: Greed Sort: An optimal sorting algorithm for multiple disks. J. ACM **42**, 919–933 (1995)

13. Shah, R., Varman, P.J., Vitter, J.S.: Online algorithms for prefetching and caching on parallel disks. In: Proceedings of the ACM Symposium on Parallel Algorithms and Architectures, pp. 255–264. ACM Press, New York (2004)
14. Vitter, J.S.: External memory algorithms and data structures: Dealing with Massive Data. ACM Comput. Surv. **33**(2), 209–271 (2001) Revised version available at http://www.cs.purdue.edu/homes/jsv/Papers/Vit.IO_survey.pdf
15. Vitter, J.S., Hutchinson, D.A.: Distribution sort with randomized cycling. J. ACM. **53** (2006)
16. Vitter, J.S., Shriver, E.A.M.: Algorithms for parallel memory I: Two-level memories. Algorithmica **12**, 110–147 (1994)

Extremal Problems

▶ Max Leaf Spanning Tree
▶ Online Interval Coloring

F

Facility Location

1997; Shmoys, Tardos, Aardal

Karen Aardal[1,2], Jaroslaw Byrka[1,2], Mohammad Mahdian[3]
[1] CWI, Amsterdam, The Netherlands
[2] Department of Mathematics and Computer Science, Eindhoven University of Technology, Eindhoven, The Netherlands
[3] Yahoo! Research, Santa Clara, CA, USA

Keywords and Synonyms

Plant location; Warehouse location

Problem Definition

Facility location problems concern situations where a planner needs to determine the location of facilities intended to serve a given set of clients. The objective is usually to minimize the sum of the cost of opening the facilities and the cost of serving the clients by the facilities, subject to various constraints, such as the number and the type of clients a facility can serve. There are many variants of the facility location problem, depending on the structure of the cost function and the constraints imposed on the solution. Early references on facility location problems include Kuehn and Hamburger [35], Balinski and Wolfe [8], Manne [40], and Balinski [7]. Review works include Krarup and Pruzan [34] and Mirchandani and Francis [42]. It is interesting to notice that the algorithm that is probably one of the most effective ones to solve the uncapacitated facility location problem to optimality is the primal-dual algorithm combined with branch-and-bound due to Erlenkotter [16] dating back to 1978. His primal-dual scheme is similar to techniques used in the modern literature on approximation algorithms.

More recently, extensive research into approximation algorithms for facility location problems has been carried out. Review articles on this topic include Shmoys [49,50] and Vygen [55]. Besides its theoretical and practical importance, facility location problems provide a showcase of common techniques in the field of approximation algorithms, as many of these techniques such as linear programming rounding, primal-dual methods, and local search have been applied successfully to this family of problems. This entry defines several facility location problems, gives a few historical pointers, and lists approximation algorithms with an emphasis on the results derived in the paper by Shmoys, Tardos, and Aardal [51]. The techniques applied to the *uncapacitated facility location* (UFL) problem are discussed in some more detail.

In the UFL problem, a set $\mathcal{F}$ of n_f *facilities* and a set C of n_c *clients* (also known as *cities*, or *demand points*) are given. For every facility $i \in \mathcal{F}$, the *facility opening* cost is equal to f_i. Furthermore, for every facility $i \in \mathcal{F}$ and client $j \in C$, there is a *connection cost* c_{ij}. The objective is to open a subset of the facilities and connect each client to an open facility so that the total cost is minimized. Notice that once the set of open facilities is specified, it is optimal to connect each client to the open facility that yields smallest connection cost. Therefore, the objective is to find a set $S \subseteq \mathcal{F}$ that minimizes $\sum_{i\in S} f_i + \sum_{j\in C} \min_{i\in S}\{c_{ij}\}$. This definition and the definitions of other variants of the facility location problem in this entry assume unit demand at each client. It is straightforward to generalize these definitions to the case where each client has a given demand. The UFL problem can be formulated as the following integer program due to Balinski [7]. Let $y_i,\ i \in \mathcal{F}$ be equal to 1 if facility i is open, and equal to 0 otherwise. Let $x_{ij},\ i \in \mathcal{F},\ j \in C$ be the fraction of client j assigned to facility i.

$$\min \sum_{i\in\mathcal{F}} f_i y_i + \sum_{i\in\mathcal{F}} \sum_{j\in C} c_{ij} x_{ij} \quad (1)$$

$$\text{subject to} \sum_{i\in\mathcal{F}} x_{ij} = 1, \quad \text{for all } j \in C, \quad (2)$$

$$x_{ij} - y_i \le 0, \quad \text{for all } i \in \mathcal{F},\ j \in C \quad (3)$$

$$x \ge 0,\ y \in \{0,1\}^{n_f} \quad (4)$$

In the *linear programming (LP) relaxation* of UFL the constraint $y \in \{0,1\}^{n_f}$ is substituted by the constraint $y \in [0,1]^{n_f}$. Notice that in the uncapacitated case, it is not necessary to require $x_{ij} \in \{0,1\},\ i \in \mathcal{F},\ j \in C$ if each client has to be serviced by precisely one facility, as $0 \le x_{ij} \le 1$ by constraints (2) and (4). Moreover, if x_{ij} is not integer, then it is always possible to create an integer solution with the same cost by assigning client *j* completely to one of the facilities currently servicing *j*.

A γ-approximation algorithm is a polynomial algorithm that, in case of minimization, is guaranteed to produce a feasible solution having value *at most* γz^*, where z^* is the value of an optimal solution, and $\gamma \ge 1$. If $\gamma = 1$ the algorithm produces an optimal solution. In case of maximization, the algorithm produces a solution having value *at least* γz^*, where $0 \le \gamma \le 1$.

Hochbaum [25] developed an $O(\log n)$-approximation algorithm for UFL. By a straightforward reduction from the Set Cover problem, it can be shown that this cannot be improved unless $NP \subseteq DTIME[n^{O(\log\log n)}]$ due to a result by Feige [17]. However, if the connection costs are restricted to come from distances in a metric space, namely $c_{ij} = c_{ji} \ge 0$ for all $i \in \mathcal{F}, j \in C$ (nonnegativity and symmetry) and $c_{ij} + c_{ji'} + c_{i'j'} \ge c_{ij'}$ for all $i, i' \in \mathcal{F}, j, j' \in C$ (triangle inequality), then constant approximation guarantees can be obtained. In all results mentioned below, except for the maximization objectives, it is assumed that the costs satisfy these restrictions. If the distances between facilities and clients are Euclidean, then for some location problems approximation schemes have been obtained [5].

Variants and Related Problems

A variant of the uncapacitated facility location problem is obtained by considering the objective coefficients c_{ij} as the per unit profit of servicing client *j* from facility *i*. The *maximization version* of UFL, max-UFL is obtained by maximizing the profit minus the facility opening cost, i. e., $\max \sum_{i\in\mathcal{F}} \sum_{j\in C} c_{ij}x_{ij} - \sum_{i\in\mathcal{F}} f_i y_i$. This variant was introduced by Cornuéjols, Fisher, and Nemhauser [15].

In the *k-median problem* the facility opening cost is removed from the objective function (1) to obtain $\min \sum_{i\in M} \sum_{j\in N} c_{ij}x_{ij}$, and the constraint that no more than *k* facilities may be opened, $\sum_{i\in M} y_i \le k$, is added. In the *k-center problem* the constraint $\sum_{i\in M} y_i \le k$ is again included, and the objective function here is to minimize the maximum distance used on a link between an open facility and a client.

In the *capacitated facility location problem* a capacity constraint $\sum_{j\in C} x_{ij} \le u_i y_i$ is added for all $i \in \mathcal{F}$. Here it is important to distinguish between the *splittable* and the *unsplittable* case, and also between *hard capacities* and *soft capacities*. In the splittable case one has $x \ge 0$, allowing for a client to be serviced by multiple depots, and in the unsplittable case one requires $x \in \{0,1\}^{n_f \times n_c}$. If each facility can be opened at most once (i. e., $y_i \in \{0,1\}$), the capacities are called hard; otherwise, if the problem allows a facility *i* to be opened any number *r* of times to serve ru_i clients, the capacities are called soft.

In the *k*-level facility location problem, the following are given: a set C of clients, *k* disjoint sets $\mathcal{F}_1, \dots, \mathcal{F}_k$ of facilities, an opening cost for each facility, and connection costs between clients and facilities. The goal is to connect each client *j* through a path $i_1,\dots,i_k$ of open facilities, with $i_\ell \in \mathcal{F}_\ell$. The connection cost for this client is $c_{ji_1} + c_{i_1 i_2} + \dots + c_{i_{k-1} i_k}$. The goal is to minimize the sum of connection costs and facility opening costs.

The problems mentioned above have all been considered by Shmoys, Tardos, and Aardal [51], with the exceptions of max-UFL, and the *k*-center and *k*-median problems. The max-UFL variant is included for historical reasons, and *k*-center and *k*-median are included since they have a rich history and since they are closely related to UFL. Results on the capacitated facility location problem with hard capacities are mentioned as this, at least from the application point of view, is a more realistic model than the soft capacity version, which was treated in [51]. For *k*-level facility location, Shmoys et al. considered the case $k = 2$. Here, the problem for general *k* is considered.

There are many other variants of the facility location problem that are not discussed here. Examples include *K*-facility location [33], universal facility location [24,38], online facility location [3,18,41], fault tolerant facility location [28,30,54], facility location with outliers [12,28], multicommodity facility location [48], priority facility location [37,48], facility location with hierarchical facility costs [52], stochastic facility location [23,37,46], connected facility location [53], load-balanced facility location [22,32,37], concave-cost facility location [24], and capacitated-cable facility location [37,47].

Key Results

Many algorithms have been proposed for location problems. To begin with, a brief description of the algorithms of Shmoys, Tardos, and Aardal [51] is given. Then, a quick overview of some key results is presented. Some of the algorithms giving the best values of the approximation guarantee γ are based on solving the LP-relaxation by a polynomial algorithm, which can actually be quite time consuming, whereas some authors have suggested fast combinatorial algorithms for facility location problems with less competitive γ-values. Due to space restrictions the focus of this entry is on the algorithms that yield the best approximation guarantees. For more references the survey papers by Shmoys [49,50] and by Vygen [55] are recommended.

The Algorithms of Shmoys, Tardos, and Aardal

First the algorithm for UFL is described, and then the results that can be obtained by adaptations of the algorithm to other problems are mentioned.

The algorithm solves the LP relaxation and then, in two stages, modifies the obtained fractional solution. The first stage is called *filtering* and it is designed to bound the connection cost of each client to the most distant facility fractionally serving him. To do so, the facility opening variables y_i are scaled up by a constant and then the connection variables x_{ij} are adjusted to use the closest possible facilities.

To describe the second stage, the notion of *clustering*, formalized later by Chudak and Shmoys [13] is used. Based on the fractional solution, the instance is cut into pieces called *clusters*. Each cluster has a distinct client called the *cluster center*. This is done by iteratively choosing a client, not covered by the previous clusters, as the next cluster center, and adding to this cluster the facilities that serve the cluster center in the fractional solution, along with other clients served by these facilities. This construction of clusters guarantees that the facilities in each cluster are open to a total extent of one, and therefore after opening the facility with the smallest opening cost in each cluster, the total facility opening cost that is paid does not exceed the facility opening cost of the fractional solution. Moreover, by choosing clients for the cluster centers in a greedy fashion, the algorithm makes each cluster center the minimizer of a certain cost function among the clients in the cluster. The remaining clients in the cluster are also connected to the opened facility. The triangle inequality for connection costs is now used to bound the cost of this connection. For UFL, this filtering and rounding algorithm is a 4-approximation algorithm. Shmoys et al. also show that if the filtering step is substituted by *randomized filtering*, an approximation guarantee of 3.16 is obtained.

In the same paper, adaptations of the algorithm, with and without randomized filtering, was made to yield approximation algorithms for the soft-capacitated facility location problem, and for the 2-level uncapacitated problem. Here, the results obtained using randomized filtering are discussed.

For the problem with soft capacities two versions of the problem were considered. Both have equal capacities, i. e., $u_i = u$ for all $i \in \mathcal{F}$. In the first version, a solution is "feasible" if the y-variables either take value 0, or a value between 1 and $\gamma' \geq 1$. Note that γ' is not required to be integer, so the constructed solution is not necessarily integer. This can be interpreted as allowing for each facility i to expand to have capacity $\gamma' u$ at a cost of $\gamma' f_i$. A (γ, γ')-approximation algorithm is a polynomial algorithm that produces such a feasible solution having a total cost within a factor of γ of the true optimal cost, i. e., with $y \in \{0,1\}^{n_f}$. Shmoys et al. developed a $(5.69, 4.24)$-approximation algorithm for the splittable case of this problem, and a $(7.62, 4.29)$-approximation algorithm for the unsplittable case.

In the second soft-capacitated model, the original problem is changed to allow for the y-variables to take nonnegative integer values, which can be interpreted as allowing multiple facilities of capacity u to be opened at each location. The approximation algorithms in this case produces a solution that is feasible with respect to this modified model. It is easy to show that the approximation guarantees obtained for the previous model also hold in this case, i. e., Shmoys et al. obtained a 5.69-approximation algorithm for splittable demands and a 7.62-approximation algorithm for unsplittable demands. This latter model is the one considered in most later papers, so this is the model that is referred to in the paragraph on soft capacity results below.

UFL

The first algorithm with constant performance guarantee was the 3.16-approximation algorithm by Shmoys, Tardos, and Aardal, see above. Since then numerous improvements have been made. Guha and Khuller [19,20] proved a lower bound on approximability of 1.463, and introduced a *greedy augmentation procedure*. A series of approximation algorithms based on LP-rounding was then developed (see e. g. [10,13]). There are also greedy algorithms that only use the LP-relaxation implicitly to obtain a lower bound for a primal-dual analysis. An example is the JMS 1.61-approximation algorithm developed by Jain, Mahdian, and Saberi [29]. Some algorithms combine several techniques, like the 1.52-approximation algorithm of Mahdian, Ye, and Zhang [39], which uses the JMS algorithm and the greedy augmentation procedure. Currently, the best known approximation guarantee is 1.5 reported by Byrka [10]. It is obtained by combining a randomized LP-rounding algorithm with the greedy JMS algorithm.

max-UFL

The first constant factor approximation algorithm was derived in 1977 by Cornuéjols et al. [15] for max-UFL. They showed that opening one facility at a time in a greedy fashion, choosing the facility to open as the one with highest marginal profit, until no facility with positive marginal profit can be found, yields a $(1 - 1/e) \approx 0.632$-approximation algorithm. The current best approximation factor is 0.828 by Ageev and Sviridenko [2].

***k*-median, *k*-center**

The first constant factor approximation algorithm for the k-median problem is due to Charikar, Guha, Tardos, and Shmoys [11]. This LP-rounding algorithm has the approximation ratio of $6\frac{2}{3}$. The currently best known approximation ratio is $3 + \epsilon$ achieved by a local search heuristic of Arya, et al. [6] (see also a separate entry *k-median and Facility Location*).

The first constant factor approximation algorithm for the k-center problem was given by Hochbaum and Shmoys [26], who developed a 2-approximation algorithm. This performance guarantee is the best possible unless $P = NP$.

Capacitated Facility Location

For the soft-capacitated problem with equal capacities, the first constant factor approximation algorithms are due to Shmoys et al. [51] for both the splittable and unsplittable demand cases, see above. Recently, a 2-approximation algorithm for the soft capacitated facility location problem with unsplittable unit demands was proposed by Mahdian et al. [39]. The integrality gap of the LP relaxation for the problem is also 2. Hence, to improve the approximation guarantee one would have to develop a better lower bound on the optimal solution.

In the hard capacities version it is important to allow for splitting the demands, as otherwise even the feasibility problem becomes difficult. Suppose demands are splittable, then we may to distinguish between the equal capacity case, where $u_i = u$ for all $i \in \mathcal{F}$, and the general case. For the problem with equal capacities, a 5.83-approximation algorithm was given by Chudak and Wiliamson [14]. The first constant factor approximation algorithm, with $\gamma = 8.53 + \epsilon$, for general capacities was given by Pál, Tardos, and Wexler [44]. This was later improved by Zhang, Chen, and Ye [57] who obtained a 5.83-approximation algorithm also for general capacities.

k-level Problem

The first constant factor approximation algorithm for $k = 2$ is due to Shmoys et al. [51], with $\gamma = 3.16$. For general k, the first algorithm, having $\gamma = 3$, was proposed by Aardal, Chudak, and Shmoys [1]. For $k = 2$, Zhang [56] developed a 1.77-approximation algorithm. He also showed that the problem for $k = 3$ and $k = 4$ can be approximated by $\gamma = 2.523$ [1] and $\gamma = 2.81$ respectively.

Applications

Facility location has numerous applications in the field of operations research. See the book edited by Mirchandani and Francis [42] or the book by Nemhauser and Wolsey [43] for a survey and a description of applications of facility location in problems such as plant location and locating bank accounts. Recently, the problem has found new applications in network design problems such as placement of routers and caches [22,36], agglomeration of traffic or data [4,21], and web server replications in a content distribution network [31,45].

Open Problems

A major open question is to determine the exact approximability threshold of UFL and close the gap between the upper bound of 1.5 [10] and the lower bound of 1.463 [20]. Another important question is to find better approximation algorithms for k-median. In particular, it would be interesting to find an LP-based 2-approximation algorithm for k-median. Such an algorithm would determine the integrality gap of the natural LP relaxation of this problem, as there are simple examples that show that this gap is at least 2.

Experimental Results

Jain et al. [28] published experimental results comparing various primal-dual algorithms. A more comprehensive experimental study of several primal-dual, local search, and heuristic algorithms is performed by Hoefer [27]. A collection of data sets for UFL and several other location problems can be found in the OR-library maintained by Beasley [9].

Cross References

- ► Assignment Problem
- ► Bin Packing (hardness of Capacitated Facility Location with unsplittable demands)

[1] This value of γ deviates slightly from the value 2.51 given in the paper. The original argument contained a minor calculation error.

- ▶ Circuit Placement
- ▶ Greedy Set-Cover Algorithms (hardness of a variant of UFL, where facilities may be built at all locations with the same cost)
- ▶ Local Approximation of Covering and Packing Problems
- ▶ Local Search for K-medians and Facility Location

Recommended Reading

1. Aardal, K., Chudak, F.A., Shmoys, D.B.: A 3-approximation algorithm for the k-level uncapacitated facility location problem. Inf. Process. Lett. **72**, 161–167 (1999)
2. Ageev, A.A., Sviridenko, M.I.: An 0.828-approximation algorithm for the uncapacitated facility location problem. Discret. Appl. Math. **93**, 149–156 (1999)
3. Anagnostopoulos, A., Bent, R., Upfal, E., van Hentenryck, P.: A simple and deterministic competitive algorithm for online facility location. Inf. Comput. **194**(2), 175–202 (2004)
4. Andrews, M., Zhang, L.: The access network design problem. In: Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp. 40–49. IEEE Computer Society, Los Alamitos, CA, USA (1998)
5. Arora, S., Raghavan, P., Rao, S.: Approximation schemes for Euclidean k-medians and related problems. In: Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC), pp. 106–113. ACM, New York (1998)
6. Arya, V., Garg, N., Khandekar, R., Meyerson, A., Munagala, K., Pandit, V.: Local search heuristics for k-median and facility location problems. In: Proceedings of the 33rd Annual ACM Symposium on Theory of Computing (STOC), pp. 21–29. ACM, New York (2001)
7. Balinski, M.L.: On finding integer solutions to linear programs. In: Proceedings of the IBM Scientific Computing Symposium on Combinatorial Problems, pp. 225–248 IBM, White Plains, NY (1966)
8. Balinski, M.L., Wolfe, P.: On Benders decomposition and a plant location problem. In ARO-27. Mathematica Inc. Princeton (1963)
9. Beasley, J.E.: Operations research library. http://people.brunel.ac.uk/~mastjjb/jeb/info.html. Accessed 2008
10. Byrka, J.: An optimal bifactor approximation algorithm for the metric uncapacitated facility location problem. In: Proceedings of the 10th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX), Lecture Notes in Computer Science, vol. 4627, pp. 29–43. Springer, Berlin (2007)
11. Charikar, M., Guha, S., Tardos, E., Shmoys, D.B.: A constant-factor approximation algorithm for the k-median problem. In: Proceedings of the 31st Annual ACM Symposium on Theory of Computing (STOC), pp. 1–10. ACM, New York (1999)
12. Charikar, M., Khuller, S., Mount, D., Narasimhan, G.: Facility location with outliers. In: Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 642–651. SIAM, Philadelphia (2001)
13. Chudak, F.A., Shmoys, D.B.: Improved approximation algorithms for the uncapacitated facility location problem. SIAM J Comput. **33**(1), 1–25 (2003)
14. Chudak, F.A., Wiliamson, D.P.: Improved approximation algorithms for capacitated facility location problems. In: Proceedings of the 7th Conference on Integer Programing and Combinatorial Optimization (IPCO). Lecture Notes in Computer Science, vol. 1610, pp. 99–113. Springer, Berlin (1999)
15. Cornuéjols, G., Fisher, M.L., Nemhauser, G.L.: Location of bank accounts to optimize float: An analytic study of exact and approximate algorithms. Manag. Sci. **8**, 789–810 (1977)
16. Erlenkotter, D.: A dual-based procedure for uncapacitated facility location problems. Oper. Res. **26**, 992–1009 (1978)
17. Feige, U.: A threshold of $\ln n$ for approximating set cover. J. ACM **45**, 634–652 (1998)
18. Fotakis, D.: On the competitive ratio for online facility location. In: Proceedings of the 30th International Colloquium on Automata, Languages and Programming (ICALP). Lecture Notes in Computer Science, vol. 2719, pp. 637–652. Springer, Berlin (2003)
19. Guha, S., Khuller, S.: Greedy strikes back: Improved facility location algorithms. In: Proceedings of the 9th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 228–248. SIAM, Philadelphia (1998)
20. Guha, S., Khuller, S.: Greedy strikes back: Improved facility location algorithms. J. Algorithms **31**, 228–248 (1999)
21. Guha, S., Meyerson, A., Munagala, K.: A constant factor approximation for the single sink edge installation problem. In: Proceedings of the 33rd Annual ACM Symposium on Theory of Computing (STOC), pp. 383–388. ACM Press, New York (2001)
22. Guha, S., Meyerson, A., Munagala, K.: Hierarchical placement and network design problems. In: Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp. 603–612. IEEE Computer Society, Los Alamitos, CA, USA (2000)
23. Gupta, A., Pál, M., Ravi, R., Sinha, A.: Boosted sampling: approximation algorithms for stochastic optimization. In: Proceedings of the 36st Annual ACM Symposium on Theory of Computing (STOC), pp. 417–426. ACM, New York (2004)
24. Hajiaghayi, M., Mahdian, M., Mirrokni, V.S.: The facility location problem with general cost functions. Netw. **42**(1), 42–47 (2003)
25. Hochbaum, D.S.: Heuristics for the fixed cost median problem. Math. Program. **22**(2), 148–162 (1982)
26. Hochbaum, D.S., Shmoys, D.B.: A best possible approximation algorithm for the k-center problem. Math. Oper. Res. **10**, 180–184 (1985)
27. Hoefer, M.: Experimental comparison of heuristic and approximation algorithms for uncapacitated facility location. In: Proceedings of the 2nd International Workshop on Experimental and Efficient Algorithms (WEA). Lecture Notes in Computer Science, vol. 2647, pp. 165–178. Springer, Berlin (2003)
28. Jain, K., Mahdian, M., Markakis, E., Saberi, A., Vazirani, V.V.: Approximation algorithms for facility location via dual fitting with factor-revealing LP. J. ACM **50**(6), 795–824 (2003)
29. Jain, K., Mahdian, M., Saberi, A.: A new greedy approach for facility location problems. In: Proceedings of the 34st Annual ACM Symposium on Theory of Computing (STOC) pp. 731–740, ACM Press, New York (2002)
30. Jain, K., Vazirani, V.V.: An approximation algorithm for the fault tolerant metric facility location problem. In: Approximation Algorithms for Combinatorial Optimization, Proceedings of APPROX (2000), vol. (1913) of Lecture Notes in Computer Science, pp. 177–183. Springer, Berlin (2000)
31. Jamin, S., Jin, C., Jin, Y., Raz, D., Shavitt, Y., Zhang, L.: On the placement of internet instrumentations. In: Proceedings of the 19th Annual Joint Conference of the IEEE Computer and Com-

munications Societies (INFOCOM), vol. 1, pp. 295–304. IEEE Computer Society, Los Alamitos, CA, USA (2000)
32. Karger, D., Minkoff, M.: Building Steiner trees with incomplete global knowledge. In: Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society, pp. 613–623. Los Alamitos (2000)
33. Krarup, J., Pruzan, P.M.: Ingredients of locational analysis. In: Mirchandani, P., Francis, R. (eds.) Discrete Location Theory, pp. 1–54. Wiley, New York (1990)
34. Krarup, J., Pruzan, P.M.: The simple plant location problem: Survey and synthesis. Eur. J. Oper. Res. **12**, 38–81 (1983)
35. Kuehn, A.A., Hamburger, M.J.: A heuristic program for locating warehouses. Manag. Sci. **9**, 643–666 (1963)
36. Li, B., Golin, M., Italiano, G., Deng, X., Sohraby, K.: On the optimal placement of web proxies in the internet. In: Proceedings of the 18th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), pp. 1282–1290. IEEE Computer Society, Los Alamitos (1999)
37. Mahdian, M.: Facility Location and the Analysis of Algorithms through Factor-Revealing Programs. Ph. D. thesis, MIT, Cambridge (2004)
38. Mahdian, M., Pál, M.: Universal facility location. In: Proceedings of the 11th Annual European Symposium on Algorithms (ESA). Lecture Notes in Computer Science, vol. 2832, pp. 409–421. Springer, Berlin (2003)
39. Mahdian, M., Ye, Y., Zhang, J.: Approximation algorithms for metric facility location problems. SIAM J. Comput. **36**(2), 411–432 (2006)
40. Manne, A.S.: Plant location under economies-of-scale – decentralization and computation. Manag. Sci. **11**, 213–235 (1964)
41. Meyerson, A.: Online facility location. In: Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp. 426–431. IEEE Computer Society, Los Alamitos (2001)
42. Mirchandani, P.B., Francis, R.L.: Discrete Location Theory. Wiley, New York (1990)
43. Nemhauser, G.L., Wolsey, L.A.: Integer and Combinatorial Optimization. Wiley, New York (1990)
44. Pál, M., Tardos, E., Wexler, T.: Facility location with nonuniform hard capacities. In: Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp. 329–338. IEEE Computer Society, Los Alamitos (2001)
45. Qiu, L., Padmanabhan, V.N., Voelker, G.: On the placement of web server replicas. In: Proceedings of the 20th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), pp. 1587–1596. IEEE Computer Society, Los Alamitos (2001)
46. Ravi, R., Sinha, A.: Hedging uncertainty: Approximation algorithms for stochastic optimization problems. Math. Program. **108**(1), 97–114 (2006)
47. Ravi, R., Sinha, A.: Integrated logistics: Approximation algorithms combining facility location and network design. In: Proceedings of the 9th Conference on Integer Programming and Combinatorial Optimization (IPCO). Lecture Notes in Computer Science, vol. 2337, pp. 212–229. Springer, Berlin (2002)
48. Ravi, R., Sinha, A.: Multicommodity facility location. In: Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 342–349. SIAM, Philadelphia (2004)
49. Shmoys, D.B.: Approximation algorithms for facility location problems. In: Jansen, K., Khuller, S. (eds.) Approximation Algorithms for Combinatorial Optimization. Lecture Notes in Computer Science, vol. 1913, pp. 27–33. Springer, Berlin (2000)
50. Shmoys, D.B.: The design and analysis of approximation algorithms: Facility location as a case study. In: Thomas, R.R., Hosten, S., Lee, J. (eds) Proceedings of Symposia in Appl. Mathematics, vol. 61, pp. 85–97. AMS, Providence, RI, USA (2004)
51. Shmoys, D.B., Tardos, E., Aardal, K.: Approximation algorithms for facility location problems. In: Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC), pp. 265–274. ACM Press, New York (1997)
52. Svitkina, Z., Tardos, E.: Facility location with hierarchical facility costs. In: Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithm (SODA), pp. 153–161. SIAM, Philadelphia, PA, USA (2006)
53. Swamy, C., Kumar, A.: Primal-dual algorithms for connected facility location problems. Algorithmica **40**(4), 245–269 (2004)
54. Swamy, C., Shmoys, D.B.: Fault-tolerant facility location. In: Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 735–736. SIAM, Philadelphia (2003)
55. Vygen, J.: Approximation algorithms for facility location problems (lecture notes). Technical report No. 05950-OR, Research Institute for Discrete Mathematics, University of Bonn (2005) http://www.or.uni-bonn.de/~vygen/fl.pdf
56. Zhang, J.: Approximating the two-level facility location problem via a quasi-greedy approach. In: Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 808–817. SIAM, Philadelphia (2004). Also, Math. Program. **108**, 159–176 (2006)
57. Zhang, J., Chen, B., Ye, Y.: A multiexchange local search algorithm for the capacitated facility location problem. Math. Oper. Res. **30**(2), 389–403 (2005)

Failure Detectors

1996; Chandra, Toueg

RACHID GUERRAOUI
School of Computer and Communication Sciences, EPFL, Lausanne, Switzerland

Keywords and Synonyms

Partial synchrony; Time-outs; Failure information; Distributed oracles

Problem Definition

A distributed system is comprised of a collection of processes. The processes typically seek to achieve some common task by communicating through message passing or shared memory. Most interesting tasks require, at least at certain points of the computation, some form of *agreement* between the processes. An abstract form of such agreement is *consensus* where processes need to agree on a single value among a set of proposed values. Solving this seemingly elementary problem is at the heart of reliable distributed computing and, in particular, of distributed database commitment, total ordering of messages, and emulations of many shared object types.

Fischer, Lynch, and Paterson's seminal result in the theory of distributed computing [13] says that consensus cannot be deterministically solved in an *asynchronous* distributed system that is prone to process failures. This impossibility holds consequently for all distributed computing problems which themselves rely on consensus.

Failures and *asynchrony* are fundamental ingredients in the consensus impossibility. The impossibility holds even if only *one* process *fails*, and it does so only by *crashing*, i. e., stopping its activities. Tolerating crashes is the least one would expect from a distributed system for the goal of distribution is in general to avoid single points of failures in centralized architectures. Usually, actual distributed applications exhibit more severe failures where processes could deviate arbitrarily from the protocol assigned to them.

Asynchrony refers to the absence of assumptions on process speeds and communication delays. This absence prevents any process from distinguishing a crashed process from a correct one and this inability is precisely what leads to the consensus impossibility. In practice, however, distributed systems are not completely asynchronous: some timing assumptions can typically be made. In the best case, if precise lower and upper bounds on communication delays and process speeds are assumed, then it is easy to show that consensus and related impossibilities can be circumvented despite the crash of any number of processes [20].

Intuitively, the way that such timing assumptions circumvent asynchronous impossibilities is by providing processes with *information about failures*, typically through *time-out* (or *heart-beat*) mechanisms, usually underlying actual distributed applications. Whereas certain information about failures can indeed be obtained in distributed systems, the accuracy of such information might vary from a system to another, depending on the underlying network, the load of the application, and the mechanisms used to detect failures. A crucial problem in this context is to characterize such information, in an abstract and precise way.

Key Results

The Failure Detector Abstraction

Chandra and Toueg [5] defined the *failure detector* abstraction as a simple way to capture failure information that is needed to circumvent asynchronous impossibilities, in particular the consensus impossibility. The model considered in [5] is a message passing one where processes can fail by *crashing*. Processes that crash stop their activities and do not recover. Processes that do not crash are said to be *correct*. At least one process is supposed to be correct in every execution of the system.

Roughly speaking, a failure detector is an oracle that provides processes with information about failures. The oracle is accessed in each computation step of a process and it provides the process with a value conveying some failure information. The value is picked from some set of values, called the *range* of the failure detector. For instance, the range could be the set of subsets of processes in the system, and each subset could depict the set of processes detected to have crashed, or considered to be correct. This would correspond to the situation where the failure detector is implemented using a time-out: every process q that does not communicate within some time period with some process p, would be included in subset of processes suspected of having crashed by p.

More specifically, a failure detector is a function, D, that associates to each *failure pattern*, F, a set of *failure detector histories* $\{H_i\} = D(F)$. Both the failure pattern and the failure detector history are themselves functions.

- A failure pattern F is a function that associates to each time t, the set of processes $F(t)$ that have indeed crashed by time t. This notion assumes the existence of a global clock, outside the control of the processes, as well as a specific concept of *crash* event associated with time. A set of failure pattern is called an *environment*.
- A failure detector history H is also a function, which associates to each process p and time t, some value v from the range of failure detector values. (The range of a failure detector D is denoted R_D.) This value v is said to be output by the failure detector D at process p and time t.

Two observations are in order.

- By construction, the output of a failure detector does not depend on the computation, i. e., on the actual steps performed by the processes, on their algorithm or the input of such algorithm. The output of the failure detector depends solely on the failure pattern, namely on whether and when processes crashed.
- A failure detector might associate several histories to each failure pattern. Each history represents a suite of possible combinations of outputs for the same given failure pattern. This captures the inherent nondeterminism of a failure detection mechanism. Such a mechanism is typically itself implemented as a distributed algorithm and the variations in communication delays for instance could lead the same mechanism

to output (even slightly) different information for the same failure pattern.

To illustrate these concepts, consider two classical examples of failure detectors.

1. The *perfect* failure detector outputs a subset of processes, i. e., the range of the failure detector is the set of subsets of processes in the system. When a process q is output at some time t at a process p, then q is said to be *detected* (of having crashed) by p. The *perfect* failure detector guarantees the two following properties:
   - Every process that crashes is eventually permanently detected;
   - No correct process is ever detected.
2. The *eventually strong* failure detector outputs a subset of processes: when a process q is output at some time t at a process p, then q is said to be *suspected* (of having crashed) by p. An *eventually strong* failure detector ensures the two following properties:
   - Every process that crashes is eventually suspected;
   - Eventually, some correct process is never suspected.

The *perfect* failure detector is *reliable*: if a process q is detected, then q has crashed. An *eventually strong* failure detector is *unreliable*: there never is any guarantee that the information that is output is accurate. The use of the the term *suspected* conveys that idea. The distinction between *unreliability* and *reliability* was precisely captured in [14] for the general context where the range of the failure detector can be arbitrary.

Consensus Algorithms

Two important results were established in [5].

Theorem 1 (Chandra-Toueg [5]) *There is an algorithm that solves consensus with a perfect failure detector.*

The theorem above implicitly says that if the distributed system provides means to implement perfect failure detection, then the consensus impossibility can be circumvented, even if all but one process crashes. In fact, the result holds for any failure pattern, i. e., in any environment.

The second theorem below relates the existence of a consensus algorithm to a resilience assumption. More specifically, the theorem holds in the *majority* environment, which is the set of failure patterns where more than half of the processes are correct.

Theorem 2 (Chandra-Toueg [5]) *There is an algorithm that implements consensus with an eventually strong failure detector in the majority environment.*

The algorithm underlying the result above is similar to *eventually synchronous* consensus algorithms [10] and share also some similarities with the *Paxos* algorithm [18]. It is shown in [5] that no algorithm using solely the *eventually strong* failure detector can solve consensus without the majority assumption. (This result is generalized to any unreliable failure detector in [14].) This resilience lower bound is intuitively due to the possibility of partitions in a message passing system where at least half of the processes can crash and failure detection is unreliable. In shared memory for example, no such possibility exists and consensus can be solved with the *eventually strong* failure [19].

Failure Detector Reductions

Failure detectors can be compared. A failure detector D_2 is said to be *weaker* than a failure detector D_1 if there is an asynchronous algorithm, called a *reduction* algorithm, which, using D_1, can emulate D_2. Three remarks are important here.

- The fact that the reduction algorithm is asynchronous means that it does not use any other source of failure information, besides D_1.
- Emulating failure detector D_2 means implementing a distributed variable that mimics the output that could be provided by D_2.
- The existence of a reduction algorithm depends on environment. Hence, strictly speaking, the fact that a failure detector is weaker than another one depends on the environment under consideration.

If failure detector D_1 is weaker than D_2, and vice et versa, then D_1 and D_2 are said to be *equivalent*. Else, if D_1 is weaker than D_2 and D_2 is not weaker than D_1, then D_1 is said to be *strictly weaker* than D_2. Again, strictly speaking, these notions depend on the considered environment.

The ability to compare failure detectors help define a notion of *weakest* failure detector to solve a problem. Basically, a failure detector D is the weakest to solve a problem P if the two following properties are satisfied:

- There is an algorithm that solves P using D.
- If there is an algorithm that solves P using some failure detector D', then D is weaker than D'.

Theorem 3 (Chandra-Hadzilacos-Toueg [4]) *The eventually strong failure detector is the weakest to solve consensus in the majority environment.*

The weakest failure detector to implement consensus in any environment was later established in [8].

Applications

A Practical Perspective

The identification of the failure detector concept had an impact on the design of reliable distributed architectures.

Basically, a failure detector can be viewed as a first class service of a distributed system, at the same level as a name service or a file service. Time-out and heartbeat mechanisms can thus be hidden under the failure detector abstraction, which can then export a unified interface to higher level applications, including consensus and state machine replication algorithms [2,11,21].

Maybe more importantly, a failure detector service can encapsulate synchrony assumptions: these can be changed without impact on the rest of the applications. Minimal synchrony assumptions to devise specific failure detectors could be explored leading to interesting theoretical results [1,7,12].

A Theoretical Perspective

A second application of the failure detector concept is a theory of distributed computability. Failure detectors enable to classify problems. A problem A is *harder* (resp. *strictly harder*) than problem B if the weakest failure detector to solve B is weaker (resp. strictly weaker) than the weakest failure detector to solve A. (This notion is of course parametrized by a specific environment.)

Maybe surprisingly, the induced failure detection reduction between problems does not exactly match the classical *black-box* reduction notion. For instance, it is well known that there is no asynchronous distributed algorithm that can use a *Queue* abstraction to implement a *Compare-Swap* abstraction in a system of $n > 2$ processes where $n - 1$ can fail by crashing [15]. In this sense, a *Compare-Swap* abstraction is strictly more powerful (in a *black-box* sense) than a *Queue* abstraction. It turns out that:

Theorem 4 (Delporte-Fauconnier-Guerraoui [9]) *The weakest failure detector to solve the Queue problem is also the weakest to solve the Compare-Swap problem in a system of $n > 2$ processes where $n - 1$ can fail by crashing.*

In a sense, this theorem indicates that reducibility as induced by the failure detector notion is different from the traditional *black-box* reduction.

Open Problems

Several issues underlying the failure detector notion are still open. One such issue consists in identifying the weakest failure detector to solve the seminal *set-agreement* problem [6]: a decision task where processes need to agree on up to k values, instead of a single value as in consensus. Three independent groups of researchers [3,16,22] proved the impossibility of solving this problem in an asynchronous system with k failures, generalizing the consensus impossibility [13]. Determining the weakest failure detector to circumvent this impossibility would clearly help understand the fundamentals of failure detection reducibility.

Another interesting research direction is to relate the complexity of distributed algorithm with the underlying failure detector [17]. Clearly, failure detectors circumvents asynchronous impossibilities, but to what extend do they boost the complexity of distributed algorithms? One would of course expect the complexity of a solution to a problem to be higher if the failure detector is weaker. But to what extend?

Cross References

- ▶ Asynchronous Consensus Impossibility
- ▶ Atomic Broadcast
- ▶ Causal Order, Logical Clocks, State Machine Replication
- ▶ Linearizability

Recommended Reading

1. Aguilera, M.K., Delporte-Gallet, C., Fauconnier, H., Toueg, S.: On implementing Omega with weak reliability and synchrony assumptions. In: 22th ACM Symposium on Principles of Distributed Computing, pp. 306–314 (2003)
2. Bertier, M., Marin, O., Sens, P.: Performance analysis of a hierarchical failure detector. In: International Conference on Dependable Systems and Networks (DSN 2003), San Francisco, CA, USA, Proceedings, pp. 635–644. 22–25 June 2003
3. Boroswsky, E., Gafni E.: Generalized FLP impossibility result for t-resilient asynchronous computations. In: Proceedings of the 25th ACM Symposium on Theory of Computing, pp. 91–100, ACM Press
4. Chandra, T.D., Hadzilacos, V., Toueg, S.: The weakest failure detector for solving consensus. J. ACM **43**(4), 685–722 (1996)
5. Chandra, T.D., Toueg, S.: Unreliable failure detectors for reliable distributed systems. J. ACM **43**(2), 225–267 (1996)
6. Chauduri, S.: More choices allow more faults: Set consensus problems in totally asynchronous systems. Inf. Comput. **105**(1), 132–158 (1993)
7. Chen, W., Toueg, S., Aguilera, M.K.: On the quality of service of failure detectors. IEEE Trans. Comput. **51**(1), 13–32 (2002)
8. Delporte-Gallet, C., Fauconnier, H., Guerraoui, R.: Failure detection lower bounds on registers and consensus. In: Proceedings of the 16th International Symposium on Distributed Computing, LNCS 2508 (2002)
9. Delporte-Gallet, C., Fauconnier, H., Guerraoui, R.: Implementing atomic objects in a message passing system. Technical report, EPFL Lausanne (2005)
10. Dwork, C., Lynch, N.A., Stockmeyer, L.: Consensus in the presence of partial synchrony. J. ACM **35**(2), 288–323 (1988)
11. Felber, P., Guerraoui, R., Fayad, M.: Putting oo distributed programming to work. Commun. ACM **42**(11), 97–101 (1999)

12. Fernández, A., Jiménez, E., Raynal, M.: Eventual leader election with weak assumptions on initial knowledge, communication reliability and synchrony. In: Proc International Symposium on Dependable Systems and Networks (DSN), pp. 166–178 (2006)
13. Fischer, M.J., Lynch, N.A., Paterson, M.S.: Impossibility of distributed consensus with one faulty process. J. ACM **32**(2), 374–382 (1985)
14. Guerraoui, R.: Indulgent algorithms. In: Proceedings of the 19th Annual ACM Symposium on Principles of Distributed Computing, Portland, Oregon, USA, pp. 289–297, ACM, July 2000
15. Herlihy, M.: Wait-free synchronization. ACM Trans. Programm. Lang. Syst. **13**(1), 123–149 (1991)
16. Herlihy, M., Shavit, N.: The asynchronous computability theorem for *t*-resilient tasks. In: Proceedings of the 25th ACM Symposium on Theory of Computing, pp. 111–120, May 1993
17. Keidar, I., Rajsbaum, S.: On the cost of fault-tolerant consensus when there are no faults-a tutorial. In: Tutorial 21th ACM Symposium on Principles of Distributed Computing, July 2002
18. Lamport, L.: The Part-Time parliament. ACM Trans. Comput. Syst. **16**(2), 133–169 (1998)
19. Lo, W.-K., Hadzilacos, V.: Using failure detectors to solve consensus in asynchronous shared memory systems. In: Proceedings of the 8th International Workshop on Distributed Algorithms, LNCS 857, pp. 280–295, September 1994
20. Lynch, N.: Distributed Algorithms. Morgan Kauffman (1996)
21. Michel, R., Corentin, T.: In search of the holy grail: Looking for the weakest failure detector for wait-free set agreement. Technical Report TR 06-1811, INRIA, August 2006
22. Saks, M., Zaharoglou, F.: Wait-free *k*-set agreement is impossible: The topology of public knowledge. In: Proceedings of the 25th ACM Symposium on Theory of Computing, pp. 101–110, ACM Press, May 1993

False-Name-Proof Auction

2004; Yokoo, Sakurai, Matsubara

MAKOTO YOKOO
Information Science and Electrical Engineering,
Kyushu University,
Fukuoka, Japan

Keywords and Synonyms

False-name-proof auctions; Pseudonymous bidding; Robustness against false-name bids

Problem Definition

In Internet auctions, it is easy for a bidder to submit multiple bids under multiple identifiers (e. g., multiple e-mail addresses). If only one item/good is sold, a bidder cannot make any additional profit by using multiple bids. However, in combinatorial auctions, where multiple items/goods are sold simultaneously, submitting multiple bids under fictitious names can be profitable. A bid made under a fictitious name is called a *false-name bid*.

Here, use the same model as the GVA section. In addition, false-name bids are modeled as follows.

- Each bidder can use multiple identifiers.
- Each identifier is unique and cannot be impersonated.
- Nobody (except the owner) knows whether two identifiers belongs to the same bidder or not.

The goal is to design a *false-name-proof protocol*, i. e., a protocol in which using false-names is useless, thus bidders voluntarily refrain from using false-names.

The problems resulting from collusion have been discussed by many researchers. Compared with collusion, a false-name bid is easier to execute on the Internet since obtaining additional identifiers, such as another e-mail address, is cheap. False-name bids can be considered as a very restricted subclass of collusion.

Key Results

The Generalized Vickrey Auction (GVA) protocol is (dominant strategy) incentive compatible, i. e., for each bidder, truth-telling is a dominant strategy (a best strategy regardless of the action of other bidders) if there exists no false-name bids. However, when false-name bids are possible, truth-telling is no longer a dominant strategy, i. e., the GVA is not false-name-proof.

Here is an example, which is identical to Example 1 in the GVA section.

Example 1 Assume there are two goods a and b, and three bidders, bidder 1, 2, and 3, whose types are θ_1, θ_2, and θ_3, respectively. The evaluation value for a bundle $v(B, \theta_i)$ is determined as follows.

	$\{a\}$	$\{b\}$	$\{a, b\}$
θ_1	\$6	\$0	\$6
θ_2	\$0	\$0	\$8
θ_3	\$0	\$5	\$5

As shown in the GVA section, good a is allocated to bidder 1, and b is allocated to bidder 3. Bidder 1 pays \$3 and bidder 3 pays \$2.

Now consider another example.

Example 2 Assume there are only two bidders, bidder 1 and 2, whose types are θ_1 and θ_2, respectively. The evaluation value for a bundle $v(B, \theta_i)$ is determined as follows.

	$\{a\}$	$\{b\}$	$\{a, b\}$
θ_1	\$6	\$5	\$11
θ_2	\$0	\$0	\$8

In this case, the bidder 1 can obtains both goods, but he/she requires to pay \$8, since if bidder 1 does not participate, the social surplus would have been \$8. When bidder 1 does participate, bidder 1 takes everything and the social surplus except bidder 1 becomes 0. Thus, bidder 1 needs to pay the decreased amount of the social surplus, i. e., \$8.

However, bidder 1 can use another identifier, namely, bidder 3 and creates a situation identical to Example 1. Then, good *a* is allocated to bidder 1, and *b* is allocated to bidder 3. Bidder 1 pays \$3 and bidder 3 pays \$2. Since bidder 3 is a false-name of bidder 1, bidder 1 can obtain both goods by paying \$3 + \$2 = \$5. Thus, using a false-name is profitable for bidder 1.

The effects of false-name bids on combinatorial auctions are analyzed in [4]. The obtained results can be summarized as follows.

- As shown in the above example, the GVA protocol is not false-name-proof.
- There exists no false-name-proof combinatorial auction protocol that satisfies Pareto efficiency.
- If a surplus function of bidders satisfies a condition called *concavity*, then the GVA is guaranteed to be false-name-proof.

Also, a series of protocols that are false-name-proof in various settings have been developed: combinatorial auction protocols [2,3], multi-unit auction protocols [1], and double auction protocols [5].

Furthermore, in [2], a distinctive class of combinatorial auction protocols called a Price-oriented, Rationing-free (PORF) protocol is identified. The description of a PORF protocol can be used as a guideline for developing strategy/false-name proof protocols.

The outline of a PORF protocol is as follows:

1. For each bidder, the price of each bundle of goods is determined independently of his/her own declaration, while it depends on the declarations of other bidders. More specifically, the price of bundle (a set of goods) B for bidder i is determined by a function $p(B, \Theta_X)$, where Θ_X is a set of declared types by other bidders X.
2. Each bidder is allocated a bundle that maximizes his/her utility independently of the allocations of other bidders (i. e., rationing-free). The prices of bundles must be determined so that *allocation feasibility* is satisfied, i. e., no two bidders want the same item.

Although a PORF protocol appears to be quite different from traditional protocol descriptions, surprisingly, it is a sufficient and necessary condition for a protocol to be strategy-proof. Furthermore, if a PORF protocol satisfies the following additional condition, it is guaranteed to be false-name-proof.

Definition 1 (No Super-Additive price increase (NSA)) For any subset of bidders $S \subseteq N$ and $N' = N \setminus S$, and for $i \in S$, denote B_i as a bundle that maximizes i's utility, then $\sum_{i \in S} p(B_i, \bigcup_{j \in S \setminus \{i\}} \{\theta_j\} \cup \Theta_{N'}) \geq p(\bigcup_{i \in S} B_i, \Theta_{N'})$.

An intuitive description of this condition is that the price of buying a combination of bundles (the right side of the inequality) must be smaller than or equal to the sum of the prices for buying these bundles separately (the left side). This condition is also a necessary condition for a protocol to be false-name-proof, i. e., any false-name-proof protocol can be described as a PORF protocol that satisfies the NSA condition.

Here is a simple example of a PORF protocol that is false-name-proof. This protocol is called the Max Minimal-Bundle (M-MB) protocol [2]. To simplify the protocol description, a concept called a *minimal* bundle is introduced.

Definition 2 (minimal bundle) Bundle B is called minimal for bidder i, if for all $B' \subset B$ and $B' \neq B$, $v(B', \theta_i) < v(B, \theta_i)$ holds.

In this new protocol, the price of bundle B for bidder i is defined as follows:

- $p(B, \Theta_X) = \max_{B_j \subseteq M, j \in X} v(B_j, \theta_j)$, where $B \cap B_j \neq \emptyset$ and B_j is minimal for bidder j.

How this protocol works using Example 1 is described here. The prices for each bidder is determined as follows.

	$\{a\}$	$\{b\}$	$\{a, b\}$
bidder 1	\$8	\$8	\$8
bidder 2	\$6	\$5	\$6
bidder 3	\$8	\$8	\$8

The minimal bundle for bidder 1 is $\{a\}$, the minimal bundle for bidder 2 is $\{a, b\}$, and the minimal bundle for bidder 3 is $\{b\}$. The price of bundle $\{a\}$ for bidder 1 is equal to the largest evaluation value of conflicting bundles. In this case, the price is \$8, i. e., the evaluation value of bidder 2 for bundle $\{a, b\}$. Similarly, the price of bidder 2 for bundle $\{a, b\}$ is 6, i. e., the evaluation value of bidder 1 for bundle $\{a\}$. As a result, bundle $\{a, b\}$ is allocated to bidder 2.

It is clear that this protocol satisfies the allocation feasibility. For each good l, choose bidder j^* and bundle B^*_j that maximize $v(B_j, \theta_j)$ where $l \in B_j$ and B_j is minimal for bidder j. Then, only bidder j^* is willing to obtain a bundle that contains good l. For all other bidders, the price of a bundle that contains l is higher than (or equal to) his/her evaluation value.

Furthermore, it is clear that this protocol satisfies the NSA condition. In this pricing scheme, $p(B \cup B', \Theta_X) =$

$\max(p(B, \Theta_X), p(B', \Theta_X))$ holds for all B, B', and Θ_X. Therefore, the following formula holds

$$p\left(\bigcup_{i \in S} B_i, \Theta_X\right) = \max_{i \in S} p(B_i, \Theta_X) \leq \sum_{i \in S} p(B_i, \Theta_X)\,.$$

Furthermore, in this pricing scheme, prices increase monotonically by adding opponents, i. e., for all $X' \supseteq X$, $p(B, \Theta_{X'}) \geq p(B, \Theta_X)$ holds. Therefore, for each i, $p(B_i, \bigcup_{j \in S \setminus \{i\}}\{\theta_j\} \cup \Theta_{N'}) \geq p(B_i, \Theta_{N'})$ holds. Therefore, the NSA condition, i. e., $\sum_{i \in S} p(B_i, \bigcup_{j \in S \setminus \{i\}}\{\theta_j\} \cup \Theta_{N'}) \geq p(\bigcup_{i \in S} B_i, \Theta_{N'})$ holds.

Applications

In Internet auctions, using multiple identifiers (e. g., multiple e-mail addresses) is quite easy and identifying each participant on the Internet is virtually impossible. Combinatorial auctions have lately attracted considerable attention. When combinatorial auctions become widely used in Internet auctions, false-name-bids could be a serious problem.

Open Problems

It is shown that there exists no false-name-proof protocol that is Pareto efficient. Thus, it is inevitable to give up the efficiency to some extent. However, the theoretical lower-bound of the efficieny loss, i. e., the amount of the efficiency loss that is inevitabe for any false-name-proof protocol, is not identified yet. Also, the efficiency loss of existing false-name-proof protocols can be quite large. More efficient false-name-proof protocols in various settings are needed.

Cross References

► Generalized Vickrey Auction

Recommended Reading

1. Iwasaki, A., Yokoo, M., Terada, K.: A robust open ascending-price multi-unit auction protocol against false-name bids. Decis. Support. Syst. **39**, 23–39 (2005)
2. Yokoo, M.: The characterization of strategy/false-name proof combinatorial auction protocols: Price-oriented, rationing-free protocol. In: Proceedings of the 18th International Joint Conference on Artificial Intelligence, pp. 733–739 (2003)
3. Yokoo, M., Sakurai, Y., Matsubara, S.: Robust combinatorial auction protocol against false-name bids. Artif. Intell. **130**, 167–181 (2001)
4. Yokoo, M., Sakurai, Y., Matsubara, S.: The effect of false-name bids in combinatorial auctions: New fraud in Internet auctions. Games Econ. Behav. **46**, 174–188 (2004)
5. Yokoo, M., Sakurai, Y., Matsubara, S.: Robust double auction protocol against false-name bids. Decis. Support. Syst. **39**, 23–39 (2005)

Fast Minimal Triangulation

2005; Heggernes, Telle, Villanger

Yngve Villanger
Department of Informatics,
University of Bergen,
Bergen, Norway

Keywords and Synonyms

Minimal fill problem

Problem Definition

Minimal triangulation is the addition of an inclusion minimal set of edges to an arbitrary undirected graph, such that a chordal graph is obtained. A graph is *chordal* if every cycle of length at least 4 contains an edge between two nonconsecutive vertices of the cycle.

More formally, Let $G = (V, E)$ be a simple and undirected graph, where $n = |V|$ and $m = |E|$. A graph $H = (V, E \cup F)$, where $E \cap F = \emptyset$ is a *triangulation* of G if H is chordal, and H is a *minimal* triangulation if there exists no $F' \subset F$, such that $H' = (V, E \cup F')$ is chordal. Edges in F are called *fill edges*, and a triangulation is minimal if and only if the removal of any single fill edge results in a chordless four cycle [10].

Since minimal triangulations were first described in the mid-1970s, a variety of algorithms have been published. A complete overview of these along with different characterizations of chordal graphs and minimal triangulations can be found in the survey of Heggernes et al. [5] on minimal triangulations. Minimal triangulation algorithms can roughly be partitioned into algorithms that obtain the triangulation through elimination orderings, and those that obtain it through vertex separators. Most of these algorithms have an $O(nm)$ running time, which becomes $O(n^3)$ for dense graphs. Among those that use elimination orderings, Kratsch and Spinrad's $O(n^{2.69})$-time algorithm [8] is currently the fastest one. The fastest algorithm is an $o(n^{2.376})$-time algorithm by Heggernes et al. [5]. This algorithm is based on vertex separators, and will be discussed further in the next section. Both the algorithm of Kratsch and Spinrad [8] and the algorithm of Heggernes et al. [5] use the matrix multiplication algorithm of Cop-

```
Algorithm FMT - Fast Minimal Triangulation
Input: An arbitrary graph $G = (V, E)$.
Output: A minimal triangulation $G'$ of $G$.

Let $Q_1, Q_2$ and $Q_3$ be empty queues;   Insert $G$ into $Q_1$;   $G' = G$;
repeat
    Construct a zero matrix $M$ with a row for each vertex in $V$ (columns are added later);
    while $Q_1$ is nonempty do
        Pop a graph $H = (U, D)$ from $Q_1$;
        Call Algorithm Partition($H$) which returns a vertex subset $A \subset U$;
        Push vertex set $A$ onto $Q_3$;
        for each connected component $C$ of $H[U \setminus A]$ do
            Add a column in $M$ such that $M(v, C) = 1$ for all vertices $v \in N_H(C)$;
            if there exists a non-edge $uv$ in $H[N_H[C]]$ with $u \in C$ then
                Push $H_C = (N_H[C], D_C)$ onto $Q_2$, where $uv \notin D_C$ if $u \in C$ and $uv \notin D$;
    Compute $MM^T$;
    Add to $G'$ the edges indicated by the nonzero elements of $MM^T$;
    while $Q_3$ is nonempty do
        Pop a vertex set $A$ from $Q_3$;
        if $G'[A]$ is not complete then Push $G'[A]$ onto $Q_2$;
    Swap names of $Q_1$ and $Q_2$;
until $Q_1$ is empty
```

Fast Minimal Triangulation, Figure 1
Fast minimal triangulation algorithm

persmith and Winograd [3] to obtain an $o(n^3)$-time algorithm.

Key Results

For a vertex set $A \subset V$, the subgraph of G induced by A is $G[A] = (A, W)$, where $uv \in W$ if $u, v \in A$ and $uv \in E\})$. The closed neighborhood of A is $N[A] = U$, where $u, v \in U$ for every $uv \in E$, where $u \in A\}$ and $N(A) = N[A] \setminus A$. A is called a *clique* if $G[A]$ is a complete graph. A vertex set $S \subset V$ is called a *separator* if $G[V \setminus S]$ is disconnected, and S is called a *minimal* separator if there exists a pair of vertices $a, b \in V \setminus S$ such that a, b are contained in different connected components of $G[V \setminus S]$, and in the same connected component of $G[V \setminus S']$ for any $S' \subset S$. A vertex set $\Omega \subseteq V$ is a *potential maximal clique* if there exists no connected component of $G[V \setminus \Omega]$ that contains Ω in its neighborhood, and for every vertex pair $u, v \in \Omega$, uv is an edge or there exists a connected component of $G[V \setminus \Omega]$ that contains both u and v in its neighborhood.

From the results in [1,7], the following recursive minimal triangulation algorithm is obtained. Find a vertex set A which is either a minimal separator or a potential maximal clique. Complete $G[A]$ into a clique. Recursively for each connected component C of $G[V \setminus A]$ where $G[N[C]]$ is not a clique, find a minimal triangulation of $G[N[C]]$. An important property here is that the set of connected components of $G[V \setminus A]$ defines independent minimal triangulation problems.

The recursive algorithm just described defines a tree, where the given input graph G is the root node, and where each connected component of $G[V \setminus A]$ becomes a child of the root node defined by G. Now continue recursively for each of the subproblems defined by these connected components. A node H which is actually a subproblem of the algorithm is defined to be at *level i*, if the distance from H to the root in the tree is i. Notice that all subproblems at the same level can be triangulated independently. Let k be the number of levels. If this recursive algorithm can be completed for every subgraph at each level in $O(f(n))$ time, then this trivially provides an $O(f(n) \cdot k)$-time algorithm.

The algorithm in Fig. 1 uses queues to obtain this level-by-level approach, and matrix multiplication to complete all the vertex separators at a given level in $O(n^\alpha)$ time, where $\alpha < 2.376$ [3]. In contrast to the previously de-

```
Algorithm Partition
Input:   A graph $H = (U, D)$ (a subproblem popped from $Q_1$).
Output:  A subset $A$ of $U$ such that either $A = N[K]$ for some connected $H[K]$
         or $A$ is a potential maximal clique of $H$ (and $G'$).

Part I: defining P
Unmark all vertices of $H$;   $k = 1$;
while there exists an unmarked vertex $u$ do
    if $\mathcal{E}_{\bar{H}}(U \setminus N_H[u]) < \frac{2}{5}|\bar{E}(H)|$ then Mark $u$ as an s-vertex (stop vertex);
    else
        $C_k = \{u\}$; Mark $u$ as a c-vertex (component vertex);
        while there exists a vertex $v \in N_H[C_k]$ which is unmarked or marked as an s-vertex do
            if $\mathcal{E}_{\bar{H}}(U \setminus N_H[C_k \cup \{v\}]) \geq \frac{2}{5}|\bar{E}(H)|$ then
                $C_k = C_k \cup \{v\}$; Mark $v$ as a c-vertex (component vertex);
            else
                Mark $v$ as a p-vertex (potential maximal clique vertex); Associate $v$ with $C_k$;
        $k = k + 1$;
$P$ = the set of all p-vertices and s-vertices;

Part II: defining A
if $H[U \setminus P]$ has a full component $C$ then $A = N_H[C]$;
else if there exist two non-adjacent vertices $u, v$ such that $u$ is an s-vertex
        and $v$ is an s-vertex or a p-vertex then $A = N_H[u]$;
else if there exist two non-adjacent p-vertices $u$ and $v$, where $u$ is associated with $C_i$
        and $v$ is associated with $C_j$ and $u \notin N_H(C_j)$ and $v \notin N_H(C_i)$ then $A = N_H[C_i \cup \{u\}]$;
else $A = P$;
```

Fast Minimal Triangulation, Figure 2
Partitioning algorithm. Let $\bar{E}(H) = W$, where $uv \in W$ if $uv \notin D$ be the set of nonedges of H. Define $\mathcal{E}_{\bar{H}}(S)$ to be the sum of degrees in $\bar{H} = (U, \bar{E})$ of vertices in $S \subseteq U = V(H)$

scribed recursive algorithm, the algorithm in Fig. 1 uses a partitioning subroutine that either returns a *set* of minimal separators or a potential maximal clique.

Even though all subproblems at the same level can be solved independently they may share vertices and edges, but no nonedges (i. e., pair of vertices that are not adjacent). Since triangulation involves edge addition, the number of nonedges will decrease for each level, and the sum of nonedges for all subproblems at the same level will never exceed n^2. The partitioning algorithm in Fig. 2 exploits this fact and has an $O(n^2 - m)$ running time, which sums up to $O(n^2)$ for each level. Thus, each level in the fast minimal triangulation algorithm given in Fig. 1 can be completed in $O(n^2 + n^\alpha)$ time, where $O(n^\alpha)$ is the time needed to compute MM^T. The partitioning algorithm in Fig. 2 actually finds a set A that defines a set of minimal separators, such that no subproblem contains more than four fifths of the nonedges in the input graph. As a result, the number of levels in the fast minimal triangulation algorithm is at most $\log_{4/5}(n^2) = 2\log_{4/5}(n)$, and the running time $O(n^\alpha \log n)$ is obtained.

Applications

The first minimal triangulation algorithms were motivated by the need to find good pivotal orderings for Gaussian elimination. Finding an optimal ordering is equivalent to solving the minimum triangulation problem, which is a nondeterministic polynomial-time hard problem. Since any minimum triangulation is also a minimal triangulation, and minimal triangulations can be found in polynomial time, then the set of minimal triangulations can be a good place to search for a pivotal ordering.

Probably because of the desired goal, the first minimal triangulation algorithms were based on orderings, and produced an ordering called a minimal elimination ordering. The problem of computing a minimal triangulation has received increasing attention since then, and several new applications and characterizations related to the vertex separator properties have been published. Two of the new applications are computing the tree-width of a graph, and reconstructing evolutionary history through phylogenetic trees [6]. The new separator-based characterizations of minimal triangulations have increased the knowledge of minimal triangulations [1,7,9]. One result based on these characterizations is an algorithm that computes the treewidth of a graph in polynomial time if the number of minimal separators is polynomially bounded [2]. A second application is faster exact (exponential-time) algorithms for computing the tree-width of a graph [4].

Open Problems

The algorithm described shows that a minimal triangulation can be found in $O((n^2 + n^\alpha) \log n)$ time, where $O(n^\alpha)$ is the time required to preform an $n \times n$ binary matrix multiplication. As a result, any improved binary matrix multiplication algorithm will result in a faster algorithm for computing a minimal triangulation. An interesting question is whether or not this relation goes the other way as well. Does there exist an $O((n^2 + n^\beta) f(n))$ algorithm for binary matrix multiplication, where $O(n^\beta)$ is the time required to find a minimal triangulation and $f(n) = o(n^{\alpha-2})$ or at least $f(n) = O(n)$. A possibly simpler and related question previously asked in [8] is: Is it at least as hard to compute a minimal triangulation as to determine whether a graph contains at least one triangle? A more algorithmic question is if there exists an $O(n^2 + n^\alpha)$-time algorithm for computing a minimal triangulation.

Cross References

► Treewidth of Graphs

Recommended Reading

1. Bouchitté, V., Todinca, I.: Treewidth and minimum fill-in: Grouping the minimal separators. SIAM J. Comput. **31**, 212–232 (2001)
2. Bouchitté, V., Todinca, I.: Listing all potential maximal cliques of a graph. Theor. Comput. Sci. **276**(1–2), 17–32 (2002)
3. Coppersmith, D., Winograd, S.: Matrix multiplication via arithmetic progressions. J. Symb. Comput. **9**(3), 251–280 (1990)
4. Fomin, F.V., Kratsch, D., Todinca, I.: Exact (exponential) algorithms for treewidth and minimum fill-in. In: ICALP of LNCS, vol. 3142, pp. 568–580. Springer, Berlin (2004)
5. Heggernes, P., Telle, J.A., Villanger, Y.: Computing minimal triangulations in time $O(n^\alpha \log n) = o(n^{2.376})$. SIAM J. Discret. Math. **19**(4), 900–913 (2005)
6. Huson, D.H., Nettles, S., Warnow, T.: Obtaining highly accurate topology estimates of evolutionary trees from very short sequences. In: RECOMB, 1999, pp. 198–207
7. Kloks, T., Kratsch, D., Spinrad, J.: On treewidth and minimum fill-in of asteroidal triple-free graphs. Theor. Comput. Sci. **175**, 309–335 (1997)
8. Kratsch, D., Spinrad, J.: Minimal fill in $O(n^{2.69})$ time. Discret. Math. **306**(3), 366–371 (2006)
9. Parra, A., Scheffler, P.: Characterizations and algorithmic applications of chordal graph embeddings. Discret. Appl. Math. **79**, 171–188 (1997)
10. Rose, D., Tarjan, R.E., Lueker, G.: Algorithmic aspects of vertex elimination on graphs. SIAM J. Comput. **5**, 146–160 (1976)

Fault-Tolerant Quantum Computation

1996; Shor, Aharonov, Ben-Or, Kitaev

BEN W. REICHARDT
Department of Computer Science,
University of California, Berkeley, CA, USA

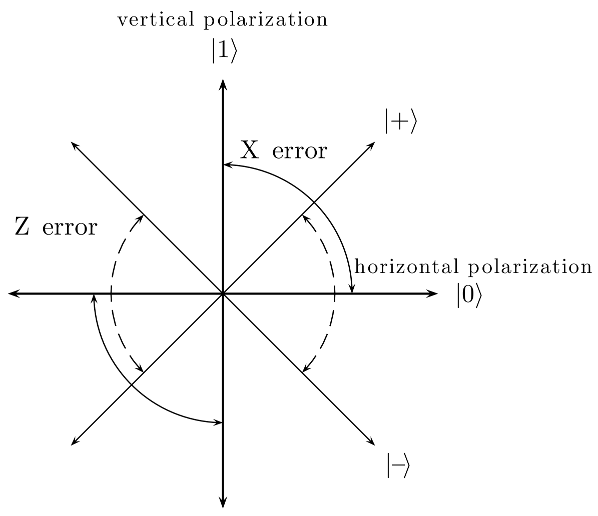
Keywords and Synonyms

Quantum noise threshold

Problem Definition

Fault tolerance is the study of reliable computation using unreliable components. With a given noise model, can one still reliably compute? For example, one can run many copies of a classical calculation in parallel, periodically using majority gates to catch and correct faults. Von Neumann showed in 1956 that if each gate fails independently with probability p, flipping its output bit $0 \leftrightarrow 1$, then such a fault-tolerance scheme still allows for arbitrarily reliable computation provided p is below some constant threshold (whose value depends on the model details) [10].

In a quantum computer, the basic gates are much more vulnerable to noise than classical transistors – after all, depending on the implementation, they are manipulating single electron spins, photon polarizations and similarly fragile subatomic particles. It might not be possible to engineer systems with noise rates less than 10^{-2}, or perhaps 10^{-3}, per gate. Additionally, the phenomenon of entanglement makes quantum systems *inherently* fragile. For example, in Schrödinger's cat state – an equal superposition between a living cat and a dead cat, often idealized as $1/\sqrt{2}(|0^n\rangle + |1^n\rangle)$ – an interaction with just one quantum bit ("qubit") can collapse, or decohere, the entire system. Fault-tolerance techniques will therefore be essential for achieving the considerable potential of quantum computers. Practical fault-tolerance techniques will need to control high noise rates and do so with low overhead, since qubits are expensive.



Fault-Tolerant Quantum Computation, Figure 1
Bit-flip X errors flip 0 and 1. In a qubit, $|0\rangle$ and $|1\rangle$ might be represented by horizontal and vertical polarization of a photon, respectively. Phase-flip Z errors flip the $\pm 45°$ polarized states $|+\rangle$ and $|-\rangle$

Quantum systems are continuous, not discrete, so there are many possible noise models. However, the essential features of quantum noise for fault-tolerance results can be captured by a simple discrete model similar to the one Von Neumann used. The main difference is that, in addition to bit-flip X errors which swap 0 and 1, there can also be phase-flip Z errors which swap $|+\rangle \equiv 1/\sqrt{2}(|0\rangle + |1\rangle)$ and $|-\rangle \equiv 1/\sqrt{2}(|0\rangle - |1\rangle)$ (Fig. 1). A noisy gate is modeled as a perfect gate followed by independent introduction of X, Z, or Y (which is both X and Z) errors with respective probabilities p_X, p_Z, p_Y. One popular model is independent depolarizing noise ($p_X = p_Z = p_Y \equiv p/3$); a depolarized qubit is completely randomized.

Faulty measurements and preparations of single-qubit states must additionally be modeled, and there can be memory noise on resting qubits. It is often assumed that measurement results can be fed into a classical computer that works perfectly and dynamically adjusts the quantum gates, although such control is not necessary. Another common, though unnecessary, assumption is that any pair of qubits in the computer can interact; this is called a *nonlocal* gate. In many proposed quantum computer implementations, however, qubit mobility is limited so gates can be applied only locally, between physically nearby qubits.

Key Results

The key result in fault tolerance is the existence of a noise *threshold*, for certain noise and computational models. The noise threshold is a positive, constant noise rate (or set of model parameters) such that with noise below this rate, reliable computation is possible. That is, given an inputless quantum circuit C of perfect gates, there exists a "simulating" circuit FTC of faulty gates such that with probability at least 2/3, say, the measured output of C agrees with that of FTC. Moreover, FTC should be only polynomially larger than C.

A quantum circuit with N gates can a priori tolerate only $O(1/N)$ error per gate, since a single failure might randomize the entire output. In 1996, Shor showed how to tolerate $O(1/\text{poly}(\log N))$ error per gate by encoding each qubit into a poly$(\log N)$-sized quantum error-correcting code; and then implementing each gate of the desired circuit directly on the encoded qubits, alternating computation and error-correction steps (similar to Von Neumann's scheme) [8]. Shor's result has two main technical pieces:

1. The discovery of quantum error-correcting codes (QECCs) was a major result. Remarkably, even though quantum errors can be continuous, codes that correct discrete errors suffice. (Measuring the syndrome of a code block projects into a discrete error event.) The first quantum code, discovered by Shor, was a nine-qubit code consisting of the concatenation of the three-qubit repetition code $|0\rangle \mapsto |000\rangle, |1\rangle \mapsto |111\rangle$ to protect against bit-flip errors, with its dual $|+\rangle \mapsto |+++\rangle, |-\rangle \mapsto |---\rangle$ to protect against phase-flip errors. Since then, many other QECCs have been discovered. Codes like the nine-qubit code that can correct bit- and phase-flip errors separately are known as Calderbank-Shor-Steane (CSS) codes, and have quantum codewords which are simultaneously superpositions over codewords of classical codes in both the $|0/1\rangle$ and $|+/-\rangle$ bases.
2. QECCs allow for quantum memory or for communicating over a noisy channel. For computation, however, it must be possible to compute on encoded states without first decoding. An operation is said to be *fault tolerant* if it cannot cause correlated errors within a code block. With the n-bit majority code, all classical gates can be applied *transversely* – an encoded gate can be implemented by applying the unencoded gate to bits i of each code block, $1 \le i \le n$. This is fault tolerant because a single failure affects at most one bit in each block, thus failures can't spread too quickly. For CSS quantum codes, the controlled-NOT gate CNOT: $|a, b\rangle \mapsto |a, a \oplus b\rangle$ can similarly be applied transversely. However, the CNOT gate by itself is not universal, so Shor also gave a fault-tolerant implementation of the Toffoli gate $|a, b, c\rangle \mapsto |a, b, c \oplus (a \wedge b)\rangle$.

Procedures are additionally needed for error correction using faulty gates, and for the initial preparation step. The encoding of $|0\rangle$ will be a highly entangled state and difficult to prepare (unlike 0^n for the classical majority code).

However, Shor did not prove the existence of a *constant* tolerable noise rate, a noise threshold. Several groups – Aharonov/Ben-Or, Kitaev, and Knill/Laflamme/Zurek – each had the idea of using smaller codes, and *concatenating* the procedure repeatedly on top of itself. Intuitively, with a distance-three code (i. e., code that corrects any one error), one expects the "effective" logical error rate of an encoded gate to be at most cp^2 for some constant c, because one error can be corrected but two errors cannot. The effective error rate for a twice-encoded gate should then be at most $c(cp^2)^2$; and since the effective error rate is dropping doubly-exponentially fast in the number of levels of concatenation, the overhead in achieving a $1/N$ error rate is only poly($\log N$). The threshold for improvement, $cp^2 < p$, is $p < 1/c$. However, this rough argument is not rigorous, because the effective error rate is ill defined, and logical errors need not fit the same model as physical errors (for example, they will not be independent).

Aharonov and Ben-Or, and Kitaev gave independent rigorous proofs of the existence of a positive constant noise threshold, in 1997 [1,5].

Broadly, there has since been progress on two fronts of the fault-tolerance problem:

1. First, work has proceeded on extending the set of noise and computation models in which a fault-tolerance threshold is known to exist. For example, correlated or even adversarial noise, leakage errors (where a qubit leaves the $|0\rangle, |1\rangle$ subspace), and non-Markovian noise (in which the environment has a memory) have all been shown to be tolerable in theory, even with only local gates.
2. Threshold existence proofs establish that building a working quantum computer is possible *in principle*. Physicists need only engineer quantum systems with a low enough constant noise rate. But realizing the potential of a quantum computer will require *practical* fault-tolerance schemes. Schemes will have to tolerate a high noise rate (not just some constant) and do so with low overhead (not just polylogarithmic). However, rough estimates of the noise rate tolerated by the original existence proofs are not promising – below 10^{-6} noise per gate. If the true threshold is only 10^{-6}, then building a quantum computer will be next to impossible. Therefore, second, there has been substantial work on optimizing fault-tolerance schemes primarily in order to improve the tolerable noise rate. These optimizations are typically evaluated with simulations and heuristic analytical models. Recently, though, Aliferis, Gottesman and Preskill have developed a method to prove reasonably good threshold lower bounds, up to 2×10^{-4}, based on counting "malignant" sets of error locations [3].

In a breakthrough, Knill has constructed a novel fault-tolerance scheme based on very efficient distance-*two* codes [6]. His codes cannot correct any errors and the scheme uses extensive postselection on no detected errors – i. e., on detecting an error, the enclosing subroutine is restarted. He has estimated a threshold above 3% per gate, an order of magnitude higher than previous estimates. Reichardt has proved a threshold lower bound of 10^{-3} for a similar scheme [7], somewhat supporting Knill's high estimate. However, reliance on postselection leads to an enormous overhead at high error rates, greatly limiting practicality. (A classical fault-tolerance scheme based on error detection could not be efficient, but quantum teleportation allows Knill's scheme to be at least theoretically efficient.) There seems to be tradeoff between the tolerable noise rate and the overhead required to achieve it.

There are several complementary approaches to quantum fault tolerance. For maximum efficiency, it is wise to exploit any known noise structure before switching to general fault-tolerance procedures. Specialized techniques include careful quantum engineering, techniques from nuclear magnetic resonance (NMR) such as dynamical decoupling and composite pulse sequences, and decoherence-free subspaces. For very small quantum computers, such techniques may give sufficient noise protection.

It is possible that an inherently reliable quantum-computing device will be engineered or discovered, like the transistor for classical computing, and this is the goal of *topological* quantum computing [4].

Applications

As quantum systems are noisy and entanglement-fragile, fault-tolerance techniques will probably be essential in implementing any quantum algorithms – including, e. g., efficient factoring and quantum simulation.

The quantum error-correcting codes originally developed for fault-tolerance have many other applications, including for example quantum key distribution.

Open Problems

Dealing with noise may turn out to be the most daunting task in building a quantum computer. Currently, physicists' low-end estimates of achievable noise rates are only slightly below theorists' high-end (mostly simulation-based) estimates of tolerable noise rates, at reasonable levels of overhead. However these estimates are made with different noise models – most simulations are based on the simple independent depolarizing noise model, and threshold lower bounds for more general noise are much lower. Also, both communities may be being too optimistic. Unanticipated noise sources may well appear as experiments progress. The probabilistic noise models used by theorists in simulations may not match reality closely enough, or the overhead/threshold tradeoff may be impractical. It is not clear if fault-tolerant quantum computing will work in practice, unless inefficiencies are wrung out of the system. Developing more efficient fault-tolerance techniques is a major open problem. Quantum system engineering, with more realistic simulations, will be required to understand better various tradeoffs and strategies for working with gate locality restrictions.

The gaps between threshold upper bounds, threshold estimates and rigorously proven threshold lower bounds are closing, at least for simple noise models. Our understanding of what to expect with more realistic noise models is less developed, though. One current line of research is in extending threshold proofs to more realistic noise models – e. g., [2]. A major open question here is whether a noise threshold can be shown to even *exist* where the bath Hamiltonian is unbounded – e. g., where system qubits are coupled to a non-Markovian, harmonic oscillator bath. Even when a threshold is known to exist, rigorous threshold lower *bounds* in more general noise models may still be far too conservative (according to arguments, mostly intuitive, known as "twirling") and, since simulations of general noise models are impractical, new ideas are needed for more efficient analyzes.

Theoretically, it is of interest what is the best asymptotic overhead in the simulating circuit FTC versus C? Overhead can be measured in terms of size N and depth/time T. With concatenated coding, the size and depth of FTC are $O(N \operatorname{poly} \log N)$ and $O(T \operatorname{poly} \log N)$, respectively. For classical circuits C, however, the depth can be only $O(T)$. It is not known if the quantum depth overhead can be improved.

Experimental Results

Fault-tolerance schemes have been simulated for large quantum systems, in order to obtain threshold estimates. For example, extensive simulations including geometric locality constraints have been run by Thaker et al. [9].

Error correction using very small codes has been experimentally verified in the lab.

URL to Code

Andrew Cross has written and distributes code for giving Monte Carlo estimates of and rigorous lower bounds on fault-tolerance thresholds: http://web.mit.edu/awcross/www/qasm-tools/. Emanuel Knill has released *Mathematica* code for estimating fault-tolerance thresholds for certain postselection-based schemes: http://arxiv.org/e-print/quant-ph/0404104.

Cross References

▶ Quantum Error Correction

Recommended Readings

1. Aharonov, D., Ben-Or, M.: Fault-tolerant quantum computation with constant error rate. In: Proc. 29th ACM Symp. on Theory of Computing (STOC), pp. 176–188, (1997). quant-ph/9906129
2. Aharonov, D., Kitaev, A.Y., Preskill, J.: Fault-tolerant quantum computation with long-range correlated noise. Phys. Rev. Lett. **96**, 050504 (2006). quant-ph/0510231
3. Aliferis, P., Gottesman, D., Preskill, J.: Quantum accuracy threshold for concatenated distance-3 codes. Quant. Inf. Comput. **6**, 97–165 (2006). quant-ph/0504218
4. Freedman, M.H., Kitaev, A.Y., Larsen, M.J., Wang, Z.: Topological quantum computation. Bull. AMS **40**(1), 31–38 (2002)
5. Kitaev, A.Y.: Quantum computations: algorithms and error correction. Russ. Math. Surv. **52**, 1191–1249 (1997)
6. Knill, E.: Quantum computing with realistically noisy devices. Nature **434**, 39–44 (2005)
7. Reichardt, B.W.: Error-detection-based quantum fault tolerance against discrete Pauli noise. Ph. D. thesis, University of California, Berkeley (2006). quant-ph/0612004
8. Shor, P.W.: Fault-tolerant quantum computation. In: Proc. 37th Symp. on Foundations of Computer Science (FOCS) (1996). quant-ph/9605011
9. Thaker, D.D., Metodi, T.S., Cross, A.W., Chuang, I.L., Chong, F.T.: Quantum memory hierarchies: Efficient designs to match available parallelism in quantum computing. In: Proc. 33rd. Int. Symp. on Computer Architecture (ISCA), pp. 378–390 (2006) quant-ph/0604070
10. von Neumann, J.: Probabilistic logic and the synthesis of reliable organisms from unreliable components. In: Shannon, C.E., McCarthy, J. (eds.) Automata Studies, pp. 43–98. Princeton University Press, Princeton (1956)

File Caching and Sharing

▶ Data Migration
▶ Online Paging and Caching
▶ P2P

Floorplan and Placement

1994; Kajitani, Nakatake, Murata, Fujiyoshi

Yoji Kajitani
Department of Information and Media Sciences,
The University of Kitakyushu, Kitakyushu, Japan

Keywords and Synonyms

Layout; Alignment; Packing; Dissection

Problem Definition

The problem is concerned with efficient coding of the constraint that defines the placement of objects on a plane without mutual overlapping. This has numerous motivations, especially in the design automation of integrated semiconductor chips, where almost hundreds of millions of rectangular modules shall be placed within a small rectangular area (chip). Until 1994, the only known coding efficient in computer aided design was *Polish-Expression* [1]. However, this can only handle a limited class of placements of the *slicing structure*. In 1994 Nakatake, Fujiyoshi, Murata, and Kajitani [2], and Murata, Fujiyoshi, Nakatake, and Kajitani [3] were finally successful to answer this long-standing problem in two contrasting ways. Their code names are *Bounded-Sliceline-Grid* (BSG) for floorplanning and *Sequence-Pair* (SP) for placement.

Notations

1. *Floorplanning, placement, compaction, packing, layout:* Often they are used as exchangeable terms. However, they have their own implications to be used in the following context. *Floorplanning* concerns the design of the plane by restricting and partitioning a given area on which objects are able to be properly *placed*. *Packing* tries a placement with an intention to reduce the area occupied by the objects. *Compaction* supports packing by pushing objects to the center of the placement. The result, including other environments, is the *layout*. BSG and SP are paired concepts, the former for "floorplanning", the latter for "placement".

2. *ABLR-relation:* The objects to be placed are assumed rectangles in this entry though they could be more general depending on the problem. For two objects p and q, p is said to be *above* q (denoted as pAq) if the bottom edge (boundary) of p is above the top edge of q. Other relations with respect to "*below*" (pBq), "*left-of*" (pLq), and "*right-of*" (pRq) are analogously defined. These four relations are generally called *ABLR-relations*. A placement without mutual overlapping of objects is said to be *feasible*. Trivially, a placement is feasible if and only if every pair of objects is in one of ABLR-relations. The example in Fig. 1 will help these definitions.

It must be noted that a pair of objects may satisfy two ABLR-relations simultaneously, but not three. Furthermore, an arbitrary set of ABLR-relations is not necessarily *consistent* for any feasible placement. For example, any set of ABLR-relations including relations (pAq), (qAr), and (rAp) is not consistent.

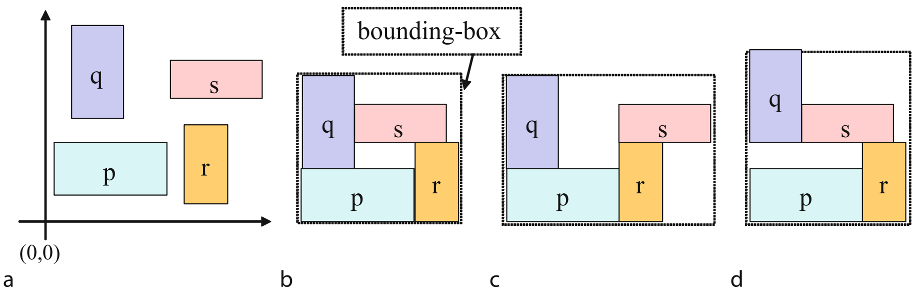
3. *Compaction:* Given a placement, its *bounding-box* is the minimum rectangle that encloses all the objects. A placement of objects is evaluated by the smallness of the bounding box's area, abbreviated as the *bb-area*. An ABLR-relation set is also evaluated by the minimum bb-area of all the placements that satisfy the set. However, given a consistent ABLR-relation set, the corresponding placement is not unique in general. Still, the minimum bb-area is easily obtained by a common technique called the "Longest-Path Algorithm". (See for example [4].)

Consider the placement whose objects are all inside the 1st quadrant of the *xy*-coordinate system, without loss of generality with respect to minimizing the bb-area. It is evident that if a given ABLR-relation set is feasible, there is an object that has no object left or below it. Place it such that its left-bottom corner is at the origin. From the remaining objects, take one that has no object left of or below it. Place it as leftward and downward as long as any ABLR-relation with already fixed objects is not violated. See Fig. 1 to catch the concept, where the ABLR-relation set is the one obtained the placement in (a) (so that it is trivially feasible). It is possible to obtain different ABLR-relation sets, according to which compaction would produce different placements.

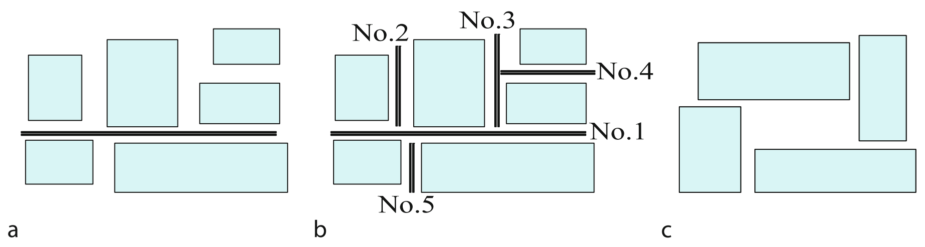
4. *Slice-line:* If it is possible to draw a straight horizontal line or vertical line to separate the objects into two groups, the line is said a *slice-line*. If each group again has a slice-line, and so does recursively, the placement is said to be a *slicing structure*. Figure 2 shows placements of slicing and non-slicing structures.

5. *Spiral:* Two structures each consisting of four line segments connected by a *T-junction* as shown in Fig. 3a are *spirals*. Their regular alignment in the 1st quadrant as shown in (b) is the *Bounded-Sliceline-Grid* or BSG. A BSG is a *floorplan*, or a *T-junction dissection*, of the rectangular area into rectangular regions called *rooms*. It is denoted as an $n \times m$ BSG if the numbers of rows and columns of its rooms are n and m, respectively. According to the left-bottom room being p-type or q-type, the BSG is said to be p-type or q-type, respectively.

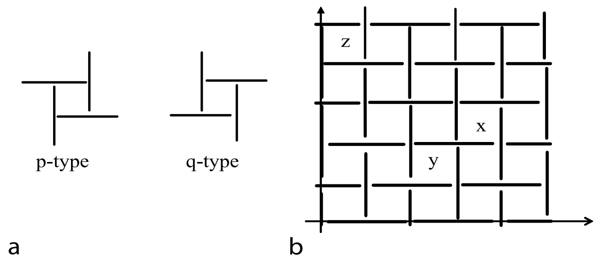
In a BSG, take two rooms x and y. The ABLR-relations between them are all that is defined by the rule: If the bottom segment of x is the top segment of y (Fig. 3), room x



Floorplan and Placement, Figure 1
a A feasible placement whose ABLR-relations could be observed differently. b Compacted placement if ABLR-relations are (qLr), (sAp), ... Its Sequence-Pair is SP = (qspr,pqrs) and Single-Sequence is SS = (2413). c Compacted placement for (qLr), (sRp), ... SP = (qpsr,pqrs). SS = (2143). d Compacted placement if (qAr), (sAp), ... SP = (qspr,prqs). SS = (3412)



Floorplan and Placement, Figure 2
a A placement with a slice-line. b A slicing structure since a slice-line can be found in each *i*th hierachy No. $k(k = 1, 2, 3, 4)$. c A placement that has no slice-line



Floorplan and Placement, Figure 3
a Two types of the spiral structure (2) 5 × 5*p*-type Bounded-Sliceline-Grid (BSG)

is above room y. Furthermore, *Transitive-Law* is assumed: If "x is above y" and "z is above x", then "z is above y".

Other relations are analogously defined.

Lemma 1 *A room is in a unique ABLR-relation with every other room.*

An $n \times n$ BSG has n^2 rooms. A BSG-assignment is a *one-to-one mapping* of n objects into the rooms of $n \times n$ BSG. ($n^2 - n$ rooms remain vacant.)

After a BSG-assignment, a pair of two objects inherits the same ABLR-relation as the ABLR-relation defined between corresponding rooms. In Fig. 3, if x, y, and z are the names of objects, are ABLR-relations among them as {(xAy), (xRz), (yBx), (yBz), (zLx), (zAy)}.

Key Results

The input is n objects that are rectangles of arbitrary sizes. The main concern is the *solution space*, the collection of distinct consistent ABLR-relation sets, to be generated by BSG or SP.

Theorem 2 ([4,5])
1) *For any feasible ABLR-relation set, there is a BSG-assignment into $n \times n$ BSG of any type that generates the same ABLR-relation set.*
2) *The size $n \times n$ is a minimum: if the number of rows or columns is less than n, there is a feasible ABLR-relation set that is not obtained by any BSG-assignment.*

The proof to 1) is not trivial [5](Appendix). The number of solutions is ${}_{n^2}C_n$. A remarkable feature of an $n \times n$ BSG is that any ABLR-relation set of n objects is generated by a proper BSG-assignment. By this property, BSG is said to be *universal* [11].

In contrast to the BSG-based generation of consistent ABLR-relation sets, SP directly imposes the ABLR-relations on objects.

A pair of permutations of object names, represented as (Γ^+, Γ^-), is called the Sequence-Pair, or SP. See Fig. 1. An SP is decoded to a unique ABLR-relation set by the rule:

Consider a pair (x, y) of names such that x is before y in Γ^-. Then (xLy) or (xAy) if x is before or after y in Γ^+, respectively. ABLR-relations "*B*" and "*R*" can be derived as the inverse of "A" and "L". Examples are given in Fig. 1.

A remarkable feature of Sequence-Pair is that its generation and decoding are both possible by simple operations. The question is what the solution space of all SP's is

Theorem 3 *Any feasible placement has a corresponding SP that generates an ABLR-relation set satisfied by the placement. On the other hand, any SP has a corresponding placement that satisfies the ABLR-relation set derived from the SP.*

Using SP, a common compaction technique mentioned before is described in a very simple way:

Minimum Area Placement from $SP = (\Gamma^+, \Gamma^-)$

1. Relabel the objects such that $\Gamma^- = (1, 2, \dots, n)$. Then $\Gamma^+ = (p_1, p_2, \dots, p_n)$ will be a permutation of numbers $1, 2, \dots, n$. It is simply a kind of normalization of *SP* [10]. But Kajitani [11] considers it a concept derived from Q-sequence [9] and studies its implication by the name of *Single-Sequence* or *SS*. In the example in Fig. 1b, p, q, r, and s are labeled as 1, 2, 3, and 4 so that *SS* = (2413).
2. Take object 1 and place it at the left-bottom corner in the 1st quadrant.
3. For $k = 2, 3, \dots, n$, place k such that its left edge is at the rightmost edge of the objects with smaller numbers than k and lie before k in *SS*, and its bottom edge is at the topmost edge of the objects with smaller numbers than k and lie after k in *SS*.

Applications

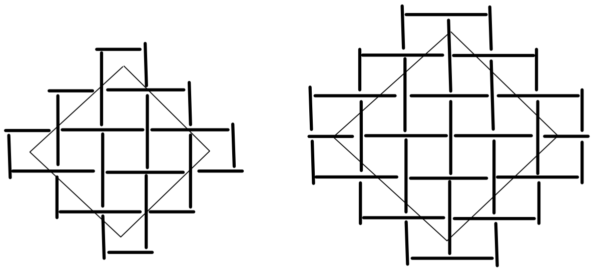
Many ideas followed after BSG and SP [2,3,4,5] as seen in the reference. They all applied a common methodology of a stochastic heuristic search, called Simulated Annealing, to generate feasible placements one after another based on some evaluation (with respect to the smallness of the bb-area), and to keep the best-so-far as the output. This methodology has become practical by the speed achieved due to their simple data structure. The first and naive implementation of BSG [2] could output the layout of sufficiently small area placement of five hundred rectangles in several minutes. (Finding a placement with the minimum bb-area is NP-hard [3].) Since then many ideas followed, including currently widely used codes such as O-tree [6], B*-tree [8], Corner–Block–List [7], Q-sequence [9], Single-Sequence [11], and others. Their common feature is in coding the nonoverlapping constraint along horizontal and vertical directions, which is the inheritant property of rectangles.

As long as applications are concerned with the rectangle placement in the minimum area, and do not mind mutual interconnection, the problem can be solved practically enough by BSG, SP, and those related ideas. However, in an integrated circuit layout problem, mutual connection is a major concern. Objects are not restricted to rectangles, even soft objects are used for performance. Many efforts have been devoted with a certain degree of success. For example, techniques concerned with rectilinear objects, rectilinear chip, insertion of small but numerous elements like buffers and decoupling capacitors, replacement for design change, symmetric placement for analog circuit design, 3-dimensional placement, etc. have been developed. Here few of them is cited but it is recommended to look at proceedings of ICCAD, DAC, ASPDAC, DATE, and journals TCAD, TCAS, particularly those that cover VLSI physical design.

Open Problems

BSG

The claim of Theorem 2 that a BSG needs n rows to provide any feasible ABLR-relation set is reasonable if considering a placement of all objects aligned vertically. This is due to the rectangular framework of a BSG. However, experiments have been suggesting a question if from the beginning [5] if we need such big BSGs. The octagonal BSG is defined in Fig. 4. It is believed to hold the following claim expecting a drastic reduction of the solution space.


Floorplan and Placement, Figure 4
Octagonal BSG of size n, p-type: a If n is odd, it has $(n^2 + 1)/2$ rooms. b If n is even, it has $(n^2 + 2n)/2$ rooms

Conjecture (BSG): For any feasible ABLR-relation set, there is an assignment of n objects into octagonal BSG of size n, any type, that generates the same ABLR-relation set.

If this is true, then the size of the solution space needed by a BSG reduces to ${}_{(n^2+1)/2}C_n$ or ${}_{(n^2+2n)/2}C_n$.

SP or SS

It is possible to define the universality of SP or SS in the same manner as defined for BSG. In general, two sequences of arbitrary k numbers $P = (p_1, p_2, \dots, p_k)$ and Q=$(q_1, q_2, \dots, q_k)$ are said *similar* with each other if $\text{ord}(p_i) = \text{ord}(q_i)$ for every i where $ord(p_i) = j$ implies that p_i is the jth smallest in the sequence. If they are single-sequences, two similar sequences generate the same set of ABLR-relations under the natural one-to-one correspondence between numbers.

An *SS* of length m (necessarily $\geq n$) is said *universal of order n* if *SS* has a subsequence (a sequence obtained from *SS* by deleting some of the numbers) that is similar to any sequence of length n. Since rooms of a BSG are considered n^2 objects, Theorem 2 implies that there is a universal *SS* of order n whose length is n^2. The known facts about smaller universal *SS* are:

1. For $n = 2, 132, 231, 213$, and 312 are the shortest universal *SS*. Note that 123 and 321 are not universal.
2. For $n = 3$, $SS = 41352$ is the shortest universal *SP*.
3. For $n = 4$, the shortest length of universal *SS* 10 or less.
4. The size of universal *SS* is $\Omega(n^2)$ [12].

Open Problem (SP)

It is still an open problem to characterize the universal *SP*. For example, give a way to 1) certify a sequence as universal and 2) generate a minimum universal sequence for general n.

Cross References

► Bin Packing
► Circuit Placement
► Slicing Floorplan Orientation
► Sphere Packing Problem

Recommended Reading

1. Wong, D.F., Liu, C.L.: A new algorithm for floorplan design. In: ACM/IEEE Design Automation Conference (DAC), November 1985, 23rd, pp. 101–107
2. Nakatake, S., Murata, H., Fujiyoshi, K., Kajitani, Y.: Bounded Sliceline Grid (BSG) for module packing. IEICE Technical Report, October 1994, VLD94-66, vol. 94, no. 313, pp. 19–24 (in Japanese)
3. Murata, H., Fujiyoshi, K., Nakatake, S., Kajitani, Y.: A solution space of size $(n!)^2$ for optimal rectangle packing. In: 8th Karuizawa Workshop on Circuits and Systems, April 1995, pp. 109–114
4. Murata, H., Nakatake, S., Fujiyoshi, K., Kajitani, Y.: VLSI Module placement based on rectangle-packing by Sequence-Pair. IEEE Trans. Comput. Aided Design (TCAD) **15**(12), 1518–1524 (1996)
5. Nakatake, S., Fujiyoshi, K., Murata, H., Kajitani, Y.: Module packing based on the BSG-structure and IC layout applications. IEEE TCAD **17**(6), 519–530 (1998)
6. Guo, P.N., Cheng, C.K., Yoshimura, T.: An O-tree representation of non-slicing floorplan and its applications. In: 36th DAC., June 1998, pp. 268–273
7. Hong, X., Dong, S., Ma, Y., Cai, Y., Cheng, C.K., Gu, J.: Corner Block List: An efficient topological representation of non-slicing floorplan. In: International Computer Aided Design (ICCAD) '00, November 2000, pp. 8–12,
8. Chang, Y.-C., Chang, Y.-W., Wu, G.-M., Wu, S.-W.: B*-trees: A new representation for non-slicing floorplans. In: 37th DAC, June 2000, pp. 458–463
9. Sakanushi, K., Kajitani, Y., Mehta, D.: The quarter-state-sequence floorplan representation. In: IEEE TCAS-I: **50**(3), 376–386 (2003)
10. Kodama, C., Fujiyoshi, K.: Selected Sequence-Pair: An efficient decodable packing representation in linear time using Sequence-Pair. In: Proc. ASP-DAC 2003, pp. 331–337
11. Kajitani, Y.: Theory of placement by Single-Sequence Realted with DAG, SP, BSG, and O-tree. In: International Symposium on Circuts and Systems, May 2006
12. Imahori, S.: Privatre communication, December 2005

Flow Time Minimization

2001; Becchetti, Leonardi, Marchetti-Spaccamela, Pruhs

LUCA BECCHETTI[1], STEFANO LEONARDI[1], ALBERTO MARCHETTI-SPACCAMELA[1], KIRK PRUHS[2]
[1] Department of Information and Computer Systems, University of Rome, Rome, Italy
[2] Computer Science, University of Pittsburgh, Pittsburgh, PA, USA

Keywords and Synonyms

Flow time: response time

Problem Definition

Shortest-job-first heuristics arise in sequencing problems, when the goal is minimizing the perceived latency of users of a multiuser or multitasking system. In this problem, the algorithm has to schedule a set of jobs on a pool of m identical machines. Each job has a release date and a processing time, and the goal is to minimize the average time spent by jobs in the system. This is normally considered a suitable measure of the quality of service provided by a system to interactive users. This optimization problem can be more formally described as follows:

Input A set of m identical machines and a set of n jobs $1, 2, \ldots, n$. Every job j has a release date r_j and a processing time p_j. In the sequel, $\mathcal{I}$ denotes the set of feasible input instances.

Goal The goal is minimizing the *average flow* (also known as *average response*) time of the jobs. Let C_j denote the time at which job j is completed by the system. The flow time or response time F_j of job j is defined by $F_j = C_j - r_j$. The goal is thus minimizing

$$\min \frac{1}{n} \sum_{j=1}^{n} F_j \, .$$

Since n is part of the input, this is equivalent to minimizing the *total* flow time, i. e. $\sum_{j=1}^{n} F_j$.

Off-line versus on-line In the *off-line setting*, the algorithm has full knowledge of the input instance. In particular, for every $j = 1, \ldots, n$, the algorithm knows r_j and p_j.

Conversely, in the *on-line setting*, at any time t, the algorithm is only aware of the set of jobs released up to time t.

In the sequel, A and OPT denote, respectively, the algorithm under consideration and the optimal, off-line policy for the problem. $A(I)$ and $OPT(I)$ denote the respective costs on a specific input instance I.

Further assumptions in the on-line case Further assumptions can be made as to the algorithm's knowledge of processing times of jobs. In particular, in this survey an important case is considered, realistic in many applications, i. e. that p_j is completely unknown to the on-line algorithms until the job eventually completes (*non-clairvoyance*) [1,3].

Performance metric In all cases, as is common in combinatorial optimization, the performance of the algorithm is measured with respect to its optimal, off-line counterpart. In a minimization problem such as those considered in this survey, the competitive ratio ρ_A is defined as:

$$\rho_A = \max_{I \in \mathcal{I}} \frac{A(I)}{OPT(I)} \, .$$

In the off-line case, ρ_A is the *approximation ratio* of the algorithm. In the on-line setting, ρ_A is known as the *competitive ratio* of A.

Preemption When *preemption* is allowed, a job that is being processed may be interrupted and resumed later after processing other jobs in the interim. As shown further, preemption is necessary to design efficient algorithms in the framework considered in this survey [5,6].

Key Results

Algorithms

Consider any job j in the instance and a time t in A's schedule, and denote by $w_j(t)$ the amount of time spent by A on job j until t. Denote by $x_j(t) = p_j - w_j(t)$ its *remaining processing time* at t.

The best known heuristic for minimizing the average flow time when preemption is allowed is *shortest remaining processing time* (SRPT). At any time t, SRPT executes a pending job j such that $x_j(t)$ is minimum. When preemption is not allowed, this heuristic translates to *shortest job first* (SJF): at the beginning of the schedule, or when a job completes, the algorithm chooses a pending job with the shortest processing time and runs it to completion.

Complexity

The problem under consideration is polynomially solvable on a single machine when preemption is allowed [9,10]. When preemption is allowed, SRPT is optimal for the single-machine case. On parallel machines, the best known upper bound for the preemptive case is achieved by SRPT, which was proven to be $O(\log \min n/m, P)$-approximate [6], P being the ratio between the largest and smallest processing times of the instance. Notice that SRPT is an on-line algorithm, so the previous result holds for the on-line case as well. The authors of [6] also prove that this lower bound is tight in the on-line case. In the off-line case, no non-constant lower bound is known when preemption is allowed.

In the non-preemptive case, no off-line algorithm can be better than $\Omega(n^{1/3-\epsilon})$-approximate, for every $\epsilon > 0$, the best upper bound being $O(\sqrt{n/m}\log(n/m))$ [6]. The upper and lower bound become $O(\sqrt{n})$ and $\Omega(n^{1/2-\epsilon})$ for the single machine case [5].

Extensions Many extensions have been proposed to the scenarios described above, in particular for the preemptive, on-line case. Most proposals concern the power of the algorithm or the knowledge of the input instance. For the former aspect, one interesting case is the one in which the algorithm is equipped with faster machines than its optimal counterpart. This aspect has been considered in [4]. There the authors prove that even a moderate increase in speed makes some very simple heuristics have performances that can be very close to the optimum.

As to the algorithm's knowledge of the input instance, an interesting case in the on-line setting, consistent with many real applications, is the non-clairvoyant case described above. This aspect has been considered in [1,3]. In particular, the authors of [1] proved that a randomized variant of the MLF heuristic described above achieves a competitive ratio that in the average is at most a polylogarithmic factor away from the optimum.

Applications

The first and traditional field of application for scheduling policies is resource assignment to processes in multitasking operating systems [11]. In particular, the use of shortest-job-like heuristics, notably the MLF heuristic, is documented in operating systems of wide use, such as UNIX and WINDOWS NT [8,11]. Their application to other domains, such as access to Web resources, has been considered more recently [2].

Open Problems

Shortest-job-first-based heuristics such as those considered in this survey have been studied in depth in the recent past. Still, some questions remain open. One concerns the off-line, parallel-machine case, where no non-constant lower bound on the approximation is known yet. As to the on-line case, there still is no tight lower bound for the non-clairvoyant case on parallel machines. The current $\Omega(\log n)$ lower bound was achieved for the single-machine case [7], and there are reasons to believe that it is below the one for the parallel case by a logarithmic factor.

Cross References

► Minimum Flow Time
► Minimum Weighted Completion Time
► Multi-level Feedback Queues
► Shortest Elapsed Time First Scheduling

Recommended Reading

1. Becchetti, L., Leonardi, S.: Nonclairvoyant scheduling to minimize the total flow time on single and parallel machines. J. ACM **51**(4), 517–539 (2004)
2. Crovella, M.E., Frangioso, R., Harchal-Balter, M.: Connection scheduling in web servers. In: Proceedings of the 2nd USENIX Symposium on Internet Technologies and Systems (USITS-99), 1999 pp. 243–254
3. Kalyanasundaram, B., Pruhs, K.: Minimizing flow time nonclairvoyantly. J. ACM **50**(4), 551–567 (2003)
4. Kalyanasundaram, B., Pruhs, K.: Speed is as powerful as clairvoyance. J. ACM **47**(4), 617–643 (2000)
5. Kellerer, H., Tautenhahn, T., Woeginger, G.J.: Approximability and nonapproximability results for minimizing total flow time on a single machine. In: Proceedings of 28th Annual ACM Symposium on the Theory of Computing (STOC '96), 1996, pp. 418–426
6. Leonardi, S., Raz, D.: Approximating total flow time on parallel machines. In: Proceedings of the Annual ACM Symposium on the Theory of Computing STOC, 1997, pp. 110–119
7. Motwani, R., Phillips, S., Torng, E.: Nonclairvoyant scheduling. Theor. Comput. Sci. **130**(1), 17–47 (1994)
8. Nutt, G.: Operating System Projects Using Windows NT. Addison-Wesley, Reading (1999)
9. Schrage, L.: A proof of the optimality of the shortest remaining processing time discipline. Oper. Res. **16**(1), 687–690 (1968)
10. Smith, D.R.: A new proof of the optimality of the shortest remaining processing time discipline. Oper. Res. **26**(1), 197–199 (1976)
11. Tanenbaum, A.S.: Modern Operating Systems. Prentice-Hall, Englewood Cliffs (1992)

Formal Methods

► Learning Automata
► Symbolic Model Checking

FPGA Technology Mapping

1992; Cong, Ding

JASON CONG[1], YUZHENG DING[2]
[1] Department of Computer Science, UCLA, Los Angeles, CA, USA
[2] Synopsys Inc., Mountain View, CA, USA

Keywords and Synonyms

Lookup-Table Mapping; LUT Mapping; FlowMap

Problem Definition

Introduction

Field Programmable Gate Array (FPGA) is a type of integrated circuit (IC) device that can be (re)programmed to implement custom logic functions. A majority of FPGA devices use lookup-table (LUT) as the basic logic element, where a LUT of K logic inputs (K-LUT) can implement any Boolean function of up to K variables. An FPGA also contains other logic elements, such as registers, programmable interconnect resources, and input/output resources [5].

The programming of an FPGA involves the transformation of a logic design into a form suitable for implementation on the target FPGA device. This generally takes multiple steps. For LUT based FPGAs, *technology mapping* is to transform a general Boolean logic network (obtained from the design specification through earlier transformations) into a functional equivalent K-LUT network that can be implemented by the target FPGA device. The objective of a technology mapping algorithm is to generate, among many possible solutions, an optimized one according to certain criteria, some of which are: timing optimization, which is to make the resultant implementation operable at faster speed; area minimization, which is to make the resultant implementation compact in size; power minimization, which is to make the resultant implementation low in power consumption. The algorithm presented here, named *FlowMap* [2], is for timing optimization; it was the first provably optimal polynomial time algorithm for technology mapping problems on general Boolean networks, and the concepts and approach it introduced has since generated numerous useful derivations and applications.

Data Representation and Preliminaries

The input data to a technology mapping algorithm for LUT based FPGA is a *general Boolean network*, which can be modeled as a direct acyclic graph $N = (V, E)$. A node $v \in V$ can either represent a logic signal source from outside of the network, in which case it has no incoming edge and is called a *primary input* (PI) node; or it can represent a *logic gate*, in which case it has incoming edge(s) from PIs and/or other gates, which are its logic input(s). If the logic output of the gate is also used outside of the network, its node is a primary output (PO), which can have no outgoing edge if it is only used outside.

If $\langle u, v\rangle \in E$, u is said to be a *fanin* of v, and v a *fanout* of u. For a node v, $input(v)$ denotes the set of its fanins; similarly for a subgraph H, $input(H)$ denotes the set of distinct nodes outside of H that are fanins of nodes in H. If there is a direct path in N from a node u to a node v, u is said to be a *predecessor* of v and v a *successor* of u. The *input network* of a node v, denoted N_v, is the subgraph containing v and all of its predecessors. A *cone* of a non-PI node v, denoted C_v, is a subgraph of N_v containing v and possibly some of its *non-PI* predecessors, such that for any node $u \in C_v$, there is a path from u to v in C_v. If $|input(C_v)| \le K$, C_v is called a *K-feasible* cone. The network N is *K-bounded* if every non-PI node has a K-feasible cone. A *cut* of a non-PI node v is a bipartition (X, X') of nodes in N_v such that X' is a cone of v; $input(X')$ is called the *cut-set* of (X, X'), and $n(X, X') = |input(X')|$ the *size* of the cut. If $n(X, X') \le K$, (X, X') is a *K-feasible* cut. The *volume* of (X, X') is $vol(X, X') = |X'|$.

A *topological order* of the nodes in the network N is a linear ordering of the nodes in which each node appears after all of its predecessors and before any of its successors. Such an order is always possible for an acyclic graph.

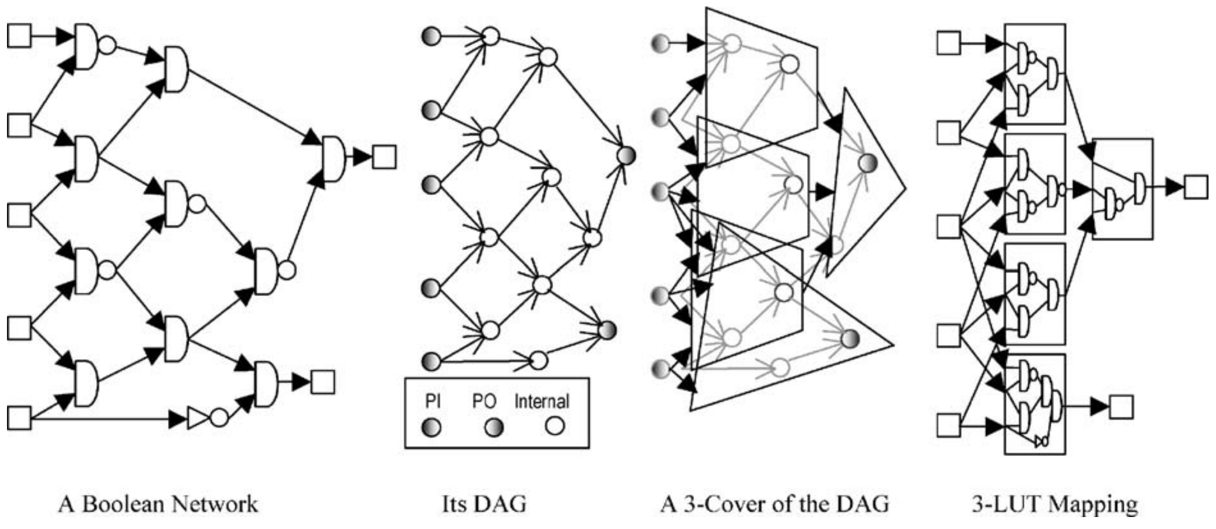
Problem Formulation

A *K-cover* of a given Boolean network N is a network $N_M = (V_M, E_M)$, where V_M consists of the PI nodes of N and some K-feasible cones of nodes in N, such that for each PO node v of N, V_M contains a cone C_v of v; and if $C_u \in V_M$, then for each non-PI node $v \in input(C_u)$, V_M also contains a cone C_v of v. edge $\langle u, C_v\rangle \in E_M$ if and only if PI node $u \in input(C_v)$; edge $\langle C_u, C_v\rangle \in E_M$ if and only if non-PI node $u \in input(C_v)$. Since each K-feasible cone can be implemented by a K-LUT, a K-cover can be implemented by a network of K-LUTs. Therefore, the *technology mapping* problem for K-LUT based FPGA, which is to transform N into a network of K-LUTs, is to find a K-cover N_M of N.

The *depth* of a network is the number of edges in its longest path. A technology mapping solution N_M is *depth optimal* if among all possible mapping solutions of N it has the minimum depth. If each level of K-LUT logic is assumed to contribute a constant amount of logic delay (known as the *unit delay* model), the minimum depth corresponds to the smallest logic propagation delay through the mapping solution, or in other words, the fastest K-LUT implementation of the network N. The problem solved by the *FlowMap* algorithm is *depth optimal technology mapping for K-LUT based FPGAs*.

A Boolean network that is not K-bounded may not have a mapping solution as defined above. To make a network K-bounded, *gate decomposition* may be used to break larger gates into smaller ones. The *FlowMap* algorithm applies, as pre-processing, an algorithm named *DMIG* [3] that converts all gates into 2-input ones in a depth optimal fashion, thus making the network K-bounded for $K \ge 2$. Different decomposition schemes may result in different K-bounded networks, and consequently different mapping solutions; the optimality of *FlowMap* is with respect to a given K-bounded network.

Figure 1 illustrates a Boolean network, its DAG, a covering with 3-feasible cones, and the resultant 3-LUT network. As illustrated, the cones in the covering may overlap; this is allowed and often beneficial. (When the mapped network is implemented, the overlapped portion of logic will be replicated into each of the K-LUTs that contain it.)



FPGA Technology Mapping, Figure 1

Key Results

The *FlowMap* algorithm takes a two-phase approach. In the first phase, it determines for each non-PI node a preferred K-feasible cone as a candidate for the covering; the cones are computed such that if used, they will yield a depth optimal mapping solution. This is the central piece of the algorithm. In the second phase the cones necessary to form a cover are chosen to generate a mapping solution.

Structure of Depth Optimal K-covers

Let $M(v)$ denote a K-cover (or equivalently, K-LUT mapping solution) of the input network N_v of v. If v is a PI, $M(v)$ consists of v itself. (For simplicity, in the rest of the article $M(v)$ shall be referred as *a K-cover of v*.) With that defined, first there is

Lemma 1 *If C_v is the K-feasible cone of v in a K-cover $M(v)$, then $M(v) = \{C_v\} + \bigcup\{M(u)\colon u \in input(C_v)\}$ where $M(u)$ is a certain K-cover of u. Conversely, if C_v is a K-feasible cone of v, and for each $u \in input(C_v)$, $M(u)$ a K-cover of u, then $M(v) = \{C_v\} + \bigcup\{M(u) : u \in input(C_v)\}$ is a K-cover of v.*

In other words, a K-cover consists of a K-feasible cone and a K-cover of each input of the cone. Note that for $u_1 \in input(C_v)$, $u_2 \in input(C_v)$, $M(u_1)$ and $M(u_2)$ may overlap, and an overlapped portion may or may not be covered the same way; the union above includes all *distinct* cones from all parts. Also note that for a given C_v, there can be different K-covers of v containing C_v, varying by the choice of $M(u)$ for each $u \in input(C_v)$.

Let $d(M(v))$ denote the depth of $M(v)$. Then

Lemma 2 *For K-cover $M(v) = \{C_v\} + \bigcup\{M(u) : u \in input(C_v)\}$, $d(M(v)) = \max\{d(M(u)) : u \in input(C_v)\}+1$.*

In particular, let $M^*(u)$ denote a K-cover of u with minimum depth, then $d(M(v)) \geq \max\{d(M^*(u)) : u \in input(C_v)\}+1$; the equality holds when every $M(u)$ in $M(v)$ is of minimum depth.

Recall that C_v defines a K-feasible cut (X, X') where $X' = C_v$, $X = N_v - C_v$. Let $H(X, X')$ denote the *height* of the cut (X, X'), defined as $H(X, X') = \max\{d(M^*(u))\colon u \in input(X')\} + 1$. Clearly, $H(X, X')$ gives the minimum depth of any K-cover of v containing $C_v = X'$. Moreover, by properly choosing the cut, $H(X, X')$ height can be minimized, which leads to a K-cover with minimum depth:

Theorem 1 *If K-feasible cut (X, X') of v has the minimum height among all K-feasible cuts of v, then the K-cover $M^*(v) = \{X'\} + \bigcup\{M^*(u) : u \in input(X')\}$, is of minimum depth among all K-covers of v.*

That is, a minimum height K-feasible cut defines a minimum depth K-cover. So the central task for depth optimal technology mapping becomes the computation of a minimum height K-feasible cut for each PO node.

By definition, the height of a cut depends on the (depths of) minimum depth K-covers of nodes in $N_v-\{v\}$. This suggests a *dynamic programming* procedure that follows topological order, so that when the minimum depth K-cover of v is to be determined, a minimum depth K-cover of each node in $N_v - \{v\}$ is already known and the height of a cut can be readily determined. This is how the first phase of the *FlowMap* algorithm is carried out.

Minimum Height K-feasible Cut Computation

The first phase of *FlowMap* was originally called the *labeling phase*, as it involves the computation of a *label* for each node in the K-bounded graph. The label of a non-PI node v, denoted $l(v)$, is defined as the minimum height of any cut of v. For convenience, the labels of PI nodes are defined to be 0.

The so defined label has an important *monotonic* property.

Lemma 3 *Let* $p = \max\{l(u) : u \in input(v)\}$, *then* $p \leq l(v) \leq p + 1$.

Note that this also implies that for any node $u \in N_v - \{v\}$, $l(u) \leq p$. Based on this, in order to find a minimum height K-feasible cut, it is sufficient to check if there is one of height p; if not, then any K-feasible cut will be of minimum height $(p + 1)$, and one always exists for a K-bounded graph.

The search for a K-feasible cut of a height p ($p > 0$; $p = 0$ is trivial) in *FlowMap* is done by transforming N_v into a *flow network* F_v and computing a *network flow* [4] on it (hence the name). The transformation is as follows. For each node $u \in N_v - \{v\}$, $l(u) < p$, F_v has two nodes u_1 and u_2, linked by a *bridge* edge $\langle u_1, u_2\rangle$; F_v has a single *sink* node t for all other nodes in N_v, and a single *source* node s. For each PI node u of N_v, which corresponds to a bridge edge $\langle u_1, u_2\rangle$ in F_v, F_v contains edge $\langle s, u_1\rangle$; for each edge $\langle u, w\rangle$ in N_v, if both u and w have bridge edges in F_v, then F_v contains edge $\langle u_2, w_1\rangle$; if u has a bridge edge but w does not, F_v contains edge $\langle u_2, t\rangle$; otherwise (neither has bridge) no corresponding edge is in F_v. The bridging edges have unit capacity; all others have infinite capacity. Noting that each edge in F_v with finite (unit) capacity corresponds to a node $u \in N_v$ with $l(u) < p$ and vice versa, and according to the Max-Flow Min-Cut Theorem [4], it can be shown

Lemma 4 *Node v has a K-feasible cut of height p if and only if F_v has a maximum network flow of size no more than K.*

On the flow network F_v, a maximum flow can be computed by running the augmenting path algorithm [4]. Once a maximum flow is obtained, the *residual graph* of the flow network is disconnected, and the corresponding *min-cut* (X, X') can be identified as follows: $v \in X'$; for $u \in N_v - \{v\}$, if it is bridged in F_v, and u_1 can be reached in a depth-first search of the residual graph from s, then $u \in X$; otherwise $u \in X'$.

Note that as soon as the flow size exceeds K, the computation can stop, knowing there will not be a desired K-feasible cut. In this case, one can modify the flow network by bridging all node in $N_v - \{v\}$ allowing the inclusion of nodes u with $l(u) = p$ in the cut computation, and find a K-feasible cut with height p+1 the same way.

An augmenting path is found in linear time to the number of edges, and there are at most K augmentations for each cut computation. Applying the algorithm to every node in topological order, one would have

Theorem 2 *In a K-bounded Boolean network of n nodes and m edges, the computation of a minimum height K-feasible cut for every node can be completed in $O(Kmn)$ time.*

The cut found by the algorithm has another property:

Lemma 5 *The cut (X, X') computed as above is the unique maximum volume min-cut; moreover, if (Y, Y') is another min-cut, then $Y' \subseteq X'$.*

Intuitively a cut of larger volume defines a larger cone which covers more logic, therefore a cut of larger volume is preferred. Note however Lemma 5 only claims maximum among min-cuts; if $n(X, X') < K$, there can be other cuts that are still K-feasible, but with larger cut size and larger cut volume. A post-processing algorithm used by *FlowMap* tries to grow (X, X') by collapsing all nodes in X', plus one or more in the cut-set, into the sink, and repeat the flow computation; this will force a cut of larger volume, an improvement if it is still K-feasible.

K-cover Construction

Once minimum height K-feasible cuts have been computed for all nodes, each node v has a K-feasible cone C_v defined by its cut, which has minimum depth. From here, constructing the K-cover $N_M = (V_M, E_M)$ is straightforward. First, the cones of all PO nodes are included in V_M. Then, for any cone $C_v \in V_M$, cone C_u for each non-PI node $u \in input(v)$ is also include in V_M; so is every PI node $u \in input(v)$. Similarly, an $\langle C_u, C_v\rangle \in E_M$ for each non-PI node $u \in input(C_v)$; $\langle u, C_v\rangle \in E_M$ for each PI node $u \in input(C_v)$.

Lemma 6 *The K-cover constructed as above is depth optimal.*

This is a linear time procedure, therefore

Theorem 3 *The problem of depth optimal technology mapping for K-LUT based FPGAs on a Boolean network of n nodes and m edges can be solved in $O(Kmn)$ time.*

Applications

The *FlowMap* algorithm has been used as a center piece or a framework for more complicated FPGA logic synthesis and technology mapping algorithms. There are many possible variations that can address various needs in its applications. Some are briefed below; details of such variations/applications can be found in [1,3].

Complicated Delay Models

With minimal change the algorithm can be applied where non-unit delay model is used, allowing delay of the nodes and/or the edges to vary, as long as they are static. Dynamic delay models, where the delay of a net is determined by its post-mapping structure, cannot be applied to the algorithm; In fact, delay optimal mapping under dynamic delay models is NP-hard [3].

Complicated Architectures

The algorithm can be adapted to FPGA architectures that are more sophisticated than homogeneous K-LUT arrays. For example, mapping for FPGA with two LUT sizes can be carried out by computing a cone for each size and dynamically choosing the best one.

Multiple Optimization Objectives

While the algorithm is for delay minimization, area minimization (in terms of the number of cones selected) as well as other objectives can also be incorporated, by adapting the criteria for cut selection. The original algorithm considers area minimization by maximizing the volume of the cuts; substantially more minimization can be achieved by considering more K-feasible cuts, and make smart choices to e. g. increase sharing among input networks, allow cuts of larger heights along no-critical paths, etc. Achieving area optimality, however, is NP-hard.

Integration with Other Optimizations

The algorithm can be combined with other types of optimizations, including retiming, logic resynthesis, and physical synthesis.

Cross References

- ▶ Circuit Partitioning: A Network-Flow-Based Balanced Min-Cut Approach
- ▶ Performance-Driven Clustering
- ▶ Sequential Circuit Technology Mapping

Recommended Reading

The *FlowMap* algorithm, with more details and experimental results, was published in [2]. General information about FPGA can be found in [5]. A good source of concepts and algorithms of network flow is [4]. Comprehensive surveys of FPGA design automation, including many variations and applications of the *FlowMap* algorithm, as well as other algorithms, are presented in [1,3].

1. Chen, D., Cong, J., Pan, P.: FPGA design automation: a survey. Foundations and Trends in Electronic Design Automation, vol 1, no 3. Now Publishers, Hanover, USA (2006)
2. Cong, J., Ding, Y.: An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs, Proc. IEEE/ACM International Conference on Computer-Aided Design, pp. 48–53. San Jose, USA (1992)
3. Cong, J., Ding, Y.: Combinational logic synthesis for LUT based field programmable gate arrays. ACM Trans. Design Autom. Electron. Sys. **1**(2): 145–204 (1996)
4. Tarjan, R.: Data Structures and Network Algorithms. SIAM. Philadelphia, USA (1983)
5. Trimberger, S.: Field-Programmable Gate Array Technology. Springer, Boston, USA (1994)

Fractional Packing and Covering Problems

1991; Plotkin, Shmoys, Tardos
1995; Plotkin, Shmoys, Tardos

GEORGE KARAKOSTAS
Department of Computing & Software,
McMaster University, Hamilton, ON, Canada

Problem Definition

This entry presents results on fast algorithms that produce approximate solutions to problems which can be formulated as Linear Programs (LP), and therefore can be solved exactly, albeit with slower running times. The general format of the family of these problems is the following: Given a set of m inequalities on n variables, and an oracle that produces the solution of an appropriate optimization problem over a convex set $P \in \mathbb{R}^n$, find a solution $x \in P$ that satisfies the inequalities, or detect that no such x exists. The basic idea of the algorithm will always be to start from an infeasible solution x, and use the optimization oracle to find a direction in which the violation of the inequalities can be decreased; this is done by calculating a vector y that is a *dual solution corresponding to* x. Then x is carefully updated towards that direction, and the process is repeated until x becomes 'approximately' feasible. In what follows, the particular problems tackled, together with the corresponding optimization oracle, as well as the different notions of 'approximation' used are defined.

- The **fractional packing problem** and its oracle are defined as follows:

  PACKING: Given an $m \times n$ matrix A, $b > 0$, and a convex set P in $\mathbb{R}^n$ such that $Ax \geq 0,\ \forall x \in P$, is there $x \in P$ such that $Ax \leq$ b?

  PACK_ORACLE: Given m-dimensional vector $y \geq 0$ and P as above, return $\bar{x} := \arg\min\{y^{\mathrm{T}}Ax : x \in P\}$.

- The **relaxed fractional packing problem** and its oracle are defined as follows:

  RELAXED PACKING: Given $\varepsilon > 0$, an $m \times n$ matrix A, $b > 0$, and convex sets P and $\hat{P}$ in $\mathbb{R}^n$ such that $P \subseteq \hat{P}$ and $Ax \geq 0,\ \forall x \in \hat{P}$, find $x \in \hat{P}$ such that $Ax \leq (1+\varepsilon)b$, or show that $\nexists x \in P$ such that $Ax \leq$ b.

  REL_PACK_ORACLE: Given m-dimensional vector $y \geq 0$ and $P, \hat{P}$ as above, return $\bar{x} \in \hat{P}$ such that $y^{\mathrm{T}}A\bar{x} \leq \min\{y^{\mathrm{T}}Ax : x \in P\}$.

- The **fractional covering problem** and its oracle are defined as follows:

  COVERING: Given an $m \times n$ matrix A, $b > 0$, and a convex set P in $\mathbb{R}^n$ such that $Ax \geq 0,\ \forall x \in P$, is there $x \in P$ such that $Ax \geq b$?

  COVER_ORACLE: Given m-dimensional vector $y \geq 0$ and P as above, return $\bar{x} := \arg\max\{y^{\mathrm{T}}Ax : x \in P\}$.

- The **simultaneous packing and covering problem** and its oracle are defined as follows:

  SIMULTANEOUS PACKING AND COVERING: Given $\hat{m} \times n$ and $(m - \hat{m}) \times n$ matrices $\hat{A}, A$ respectively, $b > 0$ and $\hat{b} > 0$, and a convex set P in $\mathbb{R}^n$ such that $Ax \geq 0$ and$\hat{A}x \geq 0,\ \forall x \in P$, is there $x \in P$ such that $Ax \leq$ b, and $\hat{A}x \geq \hat{b}$?

  SIM_ORACLE: Given P as above, a constant ν and a dual solution $(y, \hat{y})$, return $\bar{x} \in P$ such that

$$A\bar{x} \leq \nu b,\ \text{and}$$

$$y^{\mathrm{T}}A\bar{x} - \sum_{i \in I(\nu,\bar{x})} \hat{y}_i \hat{a}_i \bar{x} = \min\{y^{\mathrm{T}}Ax - \sum_{i \in I(\nu,x)} \hat{y}_i \hat{a}_i x : x \text{ a vertex of } P \text{ such that } Ax \leq \nu b\},$$

  where $I(\nu, x) := \{i : \hat{a}_i x \leq \nu b_i\}$.

- The **general problem** and its oracle are defined as follows:

  GENERAL: Given an $m \times n$ matrix A, an arbitrary vector b, and a convex set P in $\mathbb{R}^n$, is there $x \in P$ such that $Ax \leq$ b,?

  GEN_ORACLE: Given m-dimensional vector $y \geq 0$ and P as above, return $\bar{x} := \arg\min\{y^{\mathrm{T}}Ax : x \in P\}$.

Definitions and Notation

For an error parameter $\varepsilon > x0$, a point $x \in P$ is an *ε-approximation solution* for the fractional packing (or covering) problem if $Ax \leq (1+\varepsilon)b$ (or $Ax \geq (1-\varepsilon)b$). On the other hand, if $x \in P$ satisfies $Ax \leq$ b (or $Ax \geq b$), then x is an *exact solution*. For the GENERAL problem, given an error parameter $\varepsilon > 0$ and a positive tolerance vector d, $x \in P$ is an *ε-approximation solution* if $Ax \leq b + \varepsilon d$, and an *exact solution* if $Ax \leq$ b. An *ε-relaxed decision procedure* for these problems either finds an ε-approximation solution, or correctly reports that no exact solution exists. In general, for a minimization (maximization) problem, an $(1+\varepsilon)$-approximation ($(1-\varepsilon)$-*approximation*) *algorithm* returns a solution at most $(1+\varepsilon)$ (at least $(1-\varepsilon)$) times the optimal.

The algorithms developed work within time that depends polynomially on ε^{-1}, for any error parameter $\varepsilon > 0$. Their running time will also depend on the *width* ρ of the convex set P relative to the set of inequalities $Ax \leq$ b or $Ax \geq b$ defining the problem at hand. More specifically the width ρ is defined as follows for each one of the problems considered here:

- PACKING: $\rho := \max_i \max_{x \in P} \frac{a_i x}{b_i}$.
- RELAXED PACKING: $\hat{\rho} := \max_i \max_{x \in \hat{P}} \frac{a_i x}{b_i}$.
- COVERING: $\rho := \max_i \max_{x \in P} \frac{a_i x}{b_i}$.
- SIMULTANEOUS PACKING AND COVERING: $\rho := \max_{x \in P} \max\{\max_i \frac{a_i x}{b_i}, \max_i \frac{\hat{a}_i x}{\hat{b}_i}\}$.
- GENERAL: $\rho := \max_i \max_{x \in P} \frac{|a_i x - b_i|}{d_i} + 1$, where d is the tolerance vector defined above.

Key Results

Many of the results below were presented in [7] by assuming a model of computation with exact arithmetic on real numbers and exponentiation in a single step. But, as the authors mention [7], they can be converted to run on the RAM model by using approximate exponentiation, a version of the oracle that produces a *nearly* optimal solution, and a limit on the numbers used that is polynomial in the input length similar to the size of numbers used in exact linear programming algorithms. However they leave as an open problem the construction of ε-approximate solutions using polylogarithmic precision for the general case of the problems they consider (as can be done, for example, in the multicommodity flow case [4]).

Theorem 1 *For $0 < \varepsilon \leq 1$, there is a deterministic ε-relaxed decision procedure for the fractional packing problem that uses $O(\varepsilon^{-2}\rho\log(m\varepsilon^{-1}))$ calls to* PACK_ORACLE, *plus* *the time to compute Ax for the current iterate x between consecutive calls.*

For the case of P being written as a product of smaller-dimension polytopes, i. e., $P = P^1 \times \cdots \times P^k$, each P^l with width ρ^l (obviously $\rho \leq \sum_l \rho^l$), and a separate PACK_ORACLE for each P^l, A^l, then randomization can be used to potentially speed up the algorithm. By using the notation PACK_ORACLE$_l$ for the P^l, A^l oracle, the following holds:

Theorem 2 *For $0 < \varepsilon \leq 1$, there is a randomized ε-relaxed decision procedure for the fractional packing problem that is expected to use $O(\varepsilon^{-2}(\sum_l \rho^l)\log(m\varepsilon^{-1}) + k\log(\rho\varepsilon^{-1}))$ calls to PACK_ORACLE$_l$ for some $l \in \{1, \ldots, k\}$ (possibly a different l in every call), plus the time to compute $\sum_l A^l x^l$ for the current iterate $x = (x^1, x^2, \ldots, x^k)$ between consecutive calls.*

Theorem 2 holds for RELAXED PACKING as well, if ρ is replaced by $\hat{\rho}$ and PACK_ORACLE by REL_PACK_ORACLE.

In fact, one needs only an *approximate* version of PACK_ORACLE. Let $C_{\mathcal{P}}(y)$ be the minimum cost $y^{\mathrm{T}}Ax$ achieved by PACK_ORACLE for a given y.

Theorem 3 *Let PACK_ORACLE be replaced by an oracle that given vector $y \geq 0$, finds a point $\bar{x} \in P$ such that $y^T A\bar{x} \leq (1 + \varepsilon/2)C_{\mathcal{P}}(y) + (\varepsilon/2)\lambda y^T b$, where λ is minimum so that $Ax \leq \lambda b$ is satisfied by the current iterate x. Then Theorems 1 and 2 still hold.*

Theorem 3 shows that even if no efficient implementation exists for an oracle, as in, e. g., the case when this oracle solves an NP-hard problem, a fully polynomial approximation scheme for it suffices.

Similar results can be proven for the fractional covering problem (COVER_ORACLE$_l$ is defined similarly to PACK_ORACLE$_l$ above):

Theorem 4 *For $0 < \varepsilon < 1$, there is a deterministic ε-relaxed decision procedure for the fractional covering problem that uses $O(m + \rho\log^2 m + \varepsilon^{-2}\rho\log(m\varepsilon^{-1}))$ calls to COVER_ORACLE, plus the time to compute Ax for the current iterate x between consecutive calls.*

Theorem 5 *For $0 < \varepsilon < 1$, there is a randomized ε-relaxed decision procedure for the fractional packing problem that is expected to use $O(mk + (\sum_l \rho^l)\log^2 m + k\log\varepsilon^{-1} + \varepsilon^{-2}(\sum_l \rho^l)\log(m\varepsilon^{-1}))$ calls to COVER_ ORACLE$_l$ for some $l \in \{1, \ldots, k\}$ (possibly a different l in every call), plus the time to compute $\sum_l A^l x^l$ for the current iterate $x = (x^1, x^2, \ldots, x^k)$ between consecutive calls.*

Let $C_C(y)$ be the maximum cost $y^{\mathrm{T}}Ax$ achieved by COVER_ORACLE for a given y.

Theorem 6 *Let COVER_ORACLE be replaced by an oracle that given vector $y \geq 0$, finds a point $\bar{x} \in P$ such that $y^T A\bar{x} \geq (1 - \varepsilon/2)C_C(y) - (\varepsilon/2)\lambda y^T b$, where λ is maximum so that $Ax \geq \lambda b$ is satisfied by the current iterate x. Then Theorems 4 and 5 still hold.*

For the simultaneous packing and covering problem, the following is proven:

Theorem 7 *For $0 < \varepsilon \leq 1$, there is a randomized ε-relaxed decision procedure for the simultaneous packing and covering problem that is expected to use $O(m^2(\log^2 \rho)\varepsilon^{-2}\log(\varepsilon^{-1} m \log\rho))$ calls to SIM_ORACLE, and a deterministic version that uses a factor of $\log\rho$ more calls, plus the time to compute $\hat{A}x$ for the current iterate x between consecutive calls.*

For the GENERAL problem, the following is shown:

Theorem 8 *For $0 < \varepsilon < 1$, there is a deterministic ε-relaxed decision procedure for the GENERAL problem that uses $O(\varepsilon^{-2}\rho^2\log(m\rho\varepsilon^{-1}))$ calls to GEN_ORACLE, plus the time to compute Ax for the current iterate x between consecutive calls.*

The running times of these algorithms are proportional to the width ρ, and the authors devise techniques to reduce this width for many special cases of the problems considered. One example of the results obtained by these techniques is the following: If a packing problem is defined by a convex set that is a product of k smaller-dimension convex sets, i. e., $P = P^1 \times \cdots \times P^k$, and the inequalities $\sum_l A^l x^l \leq b$, then there is a randomized ε-relaxed decision procedure that is expected to use $O(\varepsilon^{-2} k\log(m\varepsilon^{-1}) + k\log k)$ calls to a subroutine that finds a minimum-cost point in $\hat{P}^l = \{x^l \in P^l : A^l x^l \leq b\}, l = 1, \ldots, k$, and a deterministic version that uses $O(\varepsilon^{-2}k^2\log(m\varepsilon^{-1}))$ such calls, plus the time to compute Ax for the current iterate x between consecutive calls. This result can be applied to the multicommodity flow problem, but the required subroutine is a single-source minimum-cost flow computation, instead of a shortest-path calculation needed for the original algorithm.

Applications

The results presented above can be used in order to obtain fast approximate solutions to linear programs, even if these can be solved exactly by LP algorithms. Many approximation algorithms are based on the rounding of the solution of such programs, and hence one might want to solve them approximately (with the overall approximation factor absorbing the LP solution approximation factor), but more efficiently. Two such examples, that appear in [7], are mentioned here.

Theorems 1, 2 can be applied for the improvement of the running time of the algorithm by Lenstra, Shmoys, and Tardos [5] for the scheduling of unrelated parallel machines without preemption ($R||C_{\max}$): N jobs are to be scheduled on M machines, with each job i scheduled on exactly one machine j with processing time p_{ij}, so that the maximum total processing time over all machines is minimized. Then, for any fixed $r > 1$, there is a deterministic $(1 + r)$-approximation algorithm that runs in $O(M^2 N \log^2 N \log M)$ time, and a randomized version that runs in $O(MN \log M \log N)$ expected time. For the version of the problem with preemption, there are polynomial-time approximation schemes that run in $O(MN^2 \log^2 N)$ time and $O(MN \log N \log M)$ expected time in the deterministic and randomized case respectively.

A well-known lower bound for the metric Traveling Salesman Problem (metric TSP) on N nodes is the Held-Karp bound [2], that can be formulated as the optimum of a linear program over the *subtour elimination polytope*. By using a randomized minimum-cut algorithm by Karger and Stein [3], one can obtain a randomized approximation scheme that computes the Held-Karp bound in $O(N^4 \log^6 N)$ expected time.

Open Problems

The main open problem is the further reduction of the running time for the approximate solution of the various fractional problems. One direction would be to improve the bounds for specific problems, as has been done very successfully for the multicommodity flow problem in a series of papers starting with Shahrokhi and Matula [8]. This same starting point also led to a series of results by Grigoriadis and Khachiyan developed independently to [7], starting with [1] which presents an algorithm with a number of calls smaller than the one in Theorem 1 by a factor of $\log(m\varepsilon^{-1})/\log m$. Considerable effort has been dedicated to the reduction of the dependence of the running time on the width of the problem or the reduction of the width itself (for example, see [9] for sequential and parallel algorithms for mixed packing and covering), so this can be another direction of improvement.

A problem left open by [7] is the development of approximation schemes for the RAM model, that use only *polylogarithmic in the input length* precision and work for the general case of the problems considered.

Cross References

▶ Minimum Makespan on Unrelated Machines

Recommended Reading

1. Grigoriadis, M.D., Khachiyan, L.G.: Fast approximation schemes for convex programs with many blocks and coupling constraints. SIAM J. Optim. **4**, 86–107 (1994)
2. Held, M., Karp, R.M.: The traveling-salesman problem and minimum cost spanning trees. Oper. Res. **18**, 1138–1162 (1970)
3. Karger, D.R., Stein, C.: An $\tilde{O}(n^2)$ algorithm for minimum cut. In: Proceeding of 25th Annual ACM Symposium on Theory of Computing (STOC), 1993, pp. 757–765
4. Leighton, F.T., Makedon, F., Plotkin, S.A., Stein, C., Tardos, É., Tragoudas, S.: Fast approximation algorithms for multicommodity flow problems. J. Comp. Syst. Sci. **50**(2), 228–243 (1995)
5. Lenstra, J.K., Shmoys, D.B., Tardos, É.: Approximation algorithms for scheduling unrelated parallel machines. Math. Program. Ser. A **24**, 259–272 (1990)
6. Plotkin, S.A., Shmoys, D.B., Tardos, É.: Fast approximation algorithms for fractional packing and covering problems. In: Proceedings of 32nd Annual IEEE Symposium on Foundations of Computer Science (FOCS), 1991, pp. 495–504
7. Plotkin, S.A., Shmoys, D.B., Tardos, É.: Fast approximation algorithms for fractional packing and covering problems. Math. Oper. Res. **20**(2) 257–301 (1995). Preliminary version appeared in [6]
8. Shahrokhi, F., Matula, D.W.: The maximum concurrent flow problem. J. ACM **37**, 318–334 (1990)
9. Young, N.E.: Sequential and parallel algorithms for mixed packing and covering. In: Proceedings of 42nd Annual IEEE Symposium on Foundations of Computer Science (FOCS), 2001, pp. 538–546

Full-Text Index Construction

▶ Suffix Array Construction
▶ Suffix Tree Construction in Hierarchical Memory
▶ Suffix Tree Construction in RAM

Fully Dynamic All Pairs Shortest Paths

2004; Demetrescu, Italiano

GIUSEPPE F. ITALIANO
Department of Information and Computer Systems, University of Rome, Rome, Italy

Problem Definition

The problem is concerned with efficiently maintaining information about all-pairs shortest paths in a dynamically changing graph. This problem has been investigated since the 60s [17,18,20], and plays a crucial role in many applications, including network optimization and routing, traffic information systems, databases, compilers, garbage collection, interactive verification systems, robotics, dataflow analysis, and document formatting.

A dynamic graph algorithm maintains a given property $\mathcal{P}$ on a graph subject to dynamic changes, such as edge insertions, edge deletions and edge weight updates. A dynamic graph algorithm should process queries on property $\mathcal{P}$ quickly, and perform update operations faster than recomputing from scratch, as carried out by the fastest static algorithm. An algorithm is said to be *fully dynamic* if it can handle both edge insertions and edge deletions. A *partially dynamic* algorithm can handle either edge insertions or edge deletions, but not both: it is *incremental* if it supports insertions only, and *decremental* if it supports deletions only. In this entry, fully dynamic algorithms for maintaining shortest paths on general directed graphs are presented.

In the *fully dynamic All Pairs Shortest Path (APSP) problem* one wishes to maintain a directed graph $G = (V, E)$ with real-valued edge weights under an intermixed sequence of the following operations:

Update(x, y, w):	update the weight of edge (x, y) to the real value w; this includes as a special case both edge insertion (if the weight is set from $+\infty$ to $w < +\infty$) and edge deletion (if the weight is set to $w = +\infty$);
Distance(x, y):	output the shortest distance from x to y.
Path(x, y):	report a shortest path from x to y, if any.

More formally, the problem can be defined as follows.

Problem 1 (Fully Dynamic All-Pairs Shortest Paths)
INPUT: *A weighted directed graph $G = (V, E)$, and a sequence σ of operations as defined above.*

OUTPUT: *A matrix D such entry $D[x, y]$ stores the distance from vertex x to vertex y throughout the sequence σ of operations.*

Throughout this entry, m and n denotes respectively the number of edges and vertices in G.

Demetrescu and Italiano [3] proposed a new approach to dynamic path problems based on maintaining classes of paths characterized by local properties, i. e., properties that hold for all proper subpaths, even if they may not hold for the entire paths. They showed that this approach can play a crucial role in the dynamic maintenance of shortest paths.

Key Results

Theorem 1 *The fully dynamic shortest path problem can be solved in $O(n^2 \log^3 n)$ amortized time per update during any intermixed sequence of operations. The space required is $O(mn)$.*

Using the same approach, Thorup [22] has shown how to slightly improve the running times:

Theorem 2 *The fully dynamic shoretest path problem can be solved in $O(n^2(\log n + \log^2(m/n)))$ amortized time per update during any intermixed sequence of operations. The space required is $O(mn)$.*

Applications

Dynamic shortest paths find applications in many areas, including network optimization and routing, transportation networks, traffic information systems, databases, compilers, garbage collection, interactive verification systems, robotics, dataflow analysis, and document formatting.

Open Problems

The recent work on dynamic shortest paths has raised some new and perhaps intriguing questions. First, can one reduce the space usage for dynamic shortest paths to $O(n^2)$? Second, and perhaps more importantly, can one solve efficiently fully dynamic *single-source* reachability and shortest paths on general graphs? Finally, are there any general techniques for making increase-only algorithms fully dynamic? Similar techniques have been widely exploited in the case of fully dynamic algorithms on undirected graphs [11,12,13].

Experimental Results

A thorough empirical study of the algorithms described in this entry is carried out in [4].

Data Sets

Data sets are described in [4].

Cross References

- ▶ Dynamic Trees
- ▶ Fully Dynamic Connectivity
- ▶ Fully Dynamic Higher Connectivity
- ▶ Fully Dynamic Higher Connectivity for Planar Graphs
- ▶ Fully Dynamic Minimum Spanning Trees
- ▶ Fully Dynamic Planarity Testing
- ▶ Fully Dynamic Transitive Closure

Recommended Reading

1. Ausiello, G., Italiano, G.F., Marchetti-Spaccamela, A., Nanni, U.: Incremental algorithms for minimal length paths. J. Algorithm **12**(4), 615–38 (1991)
2. Demetrescu, C.: Fully Dynamic Algorithms for Path Problems on Directed Graphs. Ph. D. thesis, Department of Computer and Systems Science, University of Rome "La Sapienza", Rome (2001)
3. Demetrescu, C., Italiano, G.F.: A new approach to dynamic all pairs shortest paths. J. Assoc. Comp. Mach. **51**(6), 968–992 (2004)
4. Demetrescu, C., Italiano, G.F.: Experimental analysis of dynamic all pairs shortest path algorithms. ACM Trans. Algorithms **2**(4), 578–601 (2006)
5. Demetrescu, C., Italiano, G.F.: Trade-offs for fully dynamic reachability on dags: Breaking through the $O(n^2)$ barrier. J. Assoc. Comp. Mach. **52**(2), 147–156 (2005)
6. Demetrescu, C., Italiano, G.F.: Fully Dynamic All Pairs Shortest Paths with Real Edge Weights. J. Comp. Syst. Sci. **72**(5), 813–837 (2006)
7. Even, S., Gazit, H.: Updating distances in dynamic graphs. Method. Oper. Res. **49**, 371–387 (1985)
8. Frigioni, D., Marchetti-Spaccamela, A., Nanni, U.: Semi-dynamic algorithms for maintaining single source shortest paths trees. Algorithmica **22**(3), 250–274 (1998)
9. Frigioni, D., Marchetti-Spaccamela, A., Nanni, U.: Fully dynamic algorithms for maintaining shortest paths trees. J. Algorithm **34**, 351–381 (2000)
10. Henzinger, M., King, V.: Fully dynamic biconnectivity and transitive closure. In: Proc. 36th IEEE Symposium on Foundations of Computer Science (FOCS'95). IEEE Computer Society, pp. 664–672. Los Alamos (1995)
11. Henzinger, M., King, V.: Maintaining minimum spanning forests in dynamic graphs. SIAM J. Comp. **31**(2), 364–374 (2001)
12. Henzinger, M.R., King, V.: Randomized fully dynamic graph algorithms with polylogarithmic time per operation. J. ACM **46**(4), 502–516 (1999)
13. Holm, J., de Lichtenberg, K., Thorup, M.: Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. J. ACM **48**, 723–760 (2001)
14. King, V.: Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs. In: Proc. 40th IEEE Symposium on Foundations of Computer Science (FOCS'99). IEEE Computer Society pp. 81–99. Los Alamos (1999)
15. King, V., Sagert, G.: A fully dynamic algorithm for maintaining the transitive closure. J. Comp. Syst. Sci. **65**(1), 150–167 (2002)
16. King, V., Thorup, M.: A space saving trick for directed dynamic transitive closure and shortest path algorithms. In: Proceedings of the 7th Annual International Computing and Combinatorics Conference (COCOON). LNCS, vol. 2108, pp. 268–277. Springer, Berlin (2001)
17. Loubal, P.: A network evaluation procedure. Highway Res. Rec. **205**, 96–109 (1967)
18. Murchland, J.: The effect of increasing or decreasing the length of a single arc on all shortest distances in a graph. Technical report, LBS-TNT-26, London Business School, Transport Network Theory Unit, London (1967)
19. Ramalingam, G., Reps, T.: An incremental algorithm for a generalization of the shortest path problem. J. Algorithm **21**, 267–305 (1996)
20. Rodionov, V.: The parametric problem of shortest distances. USSR Comp. Math. Math. Phys. **8**(5), 336–343 (1968)
21. Rohnert, H.: A dynamization of the all-pairs least cost problem. In: Proc. 2nd Annual Symposium on Theoretical Aspects of Computer Science, (STACS'85). LNCS, vol. 182, pp. 279–286. Springer, Berlin (1985)
22. Thorup, M.: Fully-dynamic all-pairs shortest paths: Faster and allowing negative cycles. In: Proceedings of the 9th Scandinavian Workshop on Algorithm Theory (SWAT'04), pp. 384–396. Springer, Berlin (2004)
23. Thorup, M.: Worst-case update times for fully-dynamic all-pairs shortest paths. In: Proceedings of the 37th ACM Symposium on Theory of Computing (STOC 2005), ACM. New York (2005)

Fully Dynamic Connectivity

2001; Holm, de Lichtenberg, Thorup

VALERIE KING
Department of Computer Science, University of Victoria, Victoria, BC, Canada

Keywords and Synonyms

Incremental algorithms for graphs; Fully dynamic graph algorithm for maintaining connectivity

Problem Definition

Design a data structure for an undirected graph with a fixed set of nodes which can process queries of the form "Are nodes i and j connected?" and updates of the form "Insert edge $\{i, j\}$"; "Delete edge $\{i, j\}$." The goal is to minimize update and query times, over the worst-case sequence of queries and updates. Algorithms to solve this problem are called "fully dynamic" as opposed to "partially dynamic" since both insertions and deletions are allowed.

Key Results

Holm et al. [4] gave the first deterministic fully dynamic graph algorithm for maintaining connectivity in an undirected graph with polylogarithmic amortized time per operation, specifically, $O(\log^2 n)$ amortized cost per update operation and $O(\log n/\log\log n)$ worst-case per query, where n is the number of nodes. The basic technique is extended to maintain minimum spanning trees in $O(\log^4 n)$ amortized cost per update operation, and 2-edge connectivity and biconnectivity in $O(\log^5 n)$ amortized time per operation.

The algorithm relies on a simple novel technique for maintaining a spanning forest in a graph which enables efficient search for a replacement edge when a tree edge is deleted. This technique ensures that each nontree edge is examined no more than $\log_2 n$ times. The algorithm relies on previously known tree data structures, such as top trees or ET-trees to store and quickly retrieve information about the spanning trees and the nontree edges incident to them.

Algorithms to achieve a query time $O(\log n/\log\log\log n)$ and expected amortized update time $O(\log n (\log\log n)^3)$ for connectivity and $O(\log^3 n \log\log n)$ expected amortized update time for 2-edge and biconnectivity were given in [6]. Lower bounds showing a continuum of tradeoffs for connectivity between query and update times in the cell probe model which match the known upper bounds were proved in [5]. Specifically, if t_u and t_q are the amortized update and query time, respectively, then $t_q \cdot \lg(t_u/t_q) = \Omega(\lg n)$ and $t_u \cdot \lg(t_q/t_u) = \Omega(\lg n)$.

A previously known, somewhat different randomized method for computing dynamic connectivity with $O(\log^3 n)$ amortized expected update time can be found in [2], improved to $O(\log^2 n)$ in [3]. A method which minimizes worst-case rather than amortized update time is given in [1]: $O(\sqrt{n})$ time per update for connectivity, as well as 2-edge connectivity and bipartiteness.

Open Problems

Can the worst-case update time be reduced to $o(n^{1/2})$, with polylogarithmic query time?

Can the lower bounds on the tradeoffs in [6] be matched for all possible query costs?

Applications

Dynamic connectivity has been used as a subroutine for several static graph algorithms, such as the maximum flow problem in a static graph [7], and for speeding up numerical studies of the Potts spin model.

URL to Code

See http://www.mpi-sb.mpg.de/LEDA/friends/dyngraph.html for software which implements the algorithm in [2] and other older methods.

Cross References

► Fully Dynamic All Pairs Shortest Paths
► Fully Dynamic Transitive Closure

Recommended Reading

1. Eppstein, D., Galil, Z., Italiano, G.F., Nissenzweig, A.:. Sparsification–a technique for speeding up dynamic graph algorithms. J. ACM **44**(5), 669–696.1 (1997)
2. Henzinger, M.R., King, V.: Randomized fully dynamic graph algorithms with polylogarithmic time per operation. J. ACM **46**(4), 502–536 (1999) (presented at ACM STOC 1995)
3. Henzinger, M.R., Thorup, M.: Sampling to provide or to bound: With applications to fully dynamic graph algorithms. Random Struct. Algorithms **11**(4), 369–379 (1997) (presented at ICALP 1996)
4. Holm, J., De Lichtenberg, K., Thorup, M.: Poly-logarithmic Deterministic Fully-Dynamic Algorithms for Connectivity, Minimum Spanning Tree, 2-Edge, and Biconnectivity. J. ACM **48**(4), 723–760 (2001) (presented at ACM STOC 1998)
5. Iyer, R., Karger, D., Rahul, H., Thorup, M.: An experimental study of poly-logarithmic fully-dynamic connectivity algorithms. J. Exp. Algorithmics **6**(4) (2001) (presented at ALENEX 2000)
6. Pătraşcu, M., Demaine, E.: Logarithmic Lower Bounds in the Cell-Probe Model. SIAM J. Comput. **35**(4), 932–963 (2006) (presented at ACM STOC 2004)
7. Thorup, M.: Near-optimal fully-dynamic graph connectivity. In: Proceedings of the 32th ACM Symposium on Theory of Computing pp. 343–350. ACM STOC (2000)
8. Thorup, M.: Dynamic Graph Algorithms with Applications. In: Halldórsson, M.M. (ed) 7th Scandinavian Workshop on Algorithm Theory (SWAT), Norway, 5–7 July 2000, pp. 1–9
9. Zaroliagis, C.D.: Implementations and experimental studies of dynamic graph algorithms. In: Experimental Algorithmics, Dagstuhl seminar, September 2000, Lecture Notes in Computer Science, vol. 2547. Springer (2002), Journal Article: J. Exp. Algorithmics 229–278 (2000)

Fully Dynamic Connectivity: Upper and Lower Bounds
2000; Thorup

GIUSEPPE F. ITALIANO
Department of Information and Computer Systems, University of Rome, Rome, Italy

Keywords and Synonyms

Dynamic connected components; Dynamic spanning forests

Problem Definition

The problem is concerned with efficiently maintaining information about connectivity in a dynamically changing graph. A dynamic graph algorithm maintains a given property $\mathcal{P}$ on a graph subject to dynamic changes, such as edge insertions, edge deletions and edge weight updates. A dynamic graph algorithm should process queries on property $\mathcal{P}$ quickly, and perform update operations faster than recomputing from scratch, as carried out by the fastest static algorithm. An algorithm is said to be *fully dynamic* if it can handle both edge insertions and edge deletions. A *partially dynamic* algorithm can handle either edge insertions or edge deletions, but not both: it is *incremental* if it supports insertions only, and *decremental* if it supports deletions only.

In the fully dynamic connectivity problem, one wishes to maintain an undirected graph $G = (V, E)$ under an intermixed sequence of the following operations:

***Connected*(u, v):** Return *true* if vertices u and v are in the same connected component of the graph. Return *false* otherwise.

***Insert*(x, y):** Insert a new edge between the two vertices x and y.

***Delete*(x, y):** Delete the edge between the two vertices x and y.

Key Results

In this section, a high level description of the algorithm for the fully dynamic connectivity problem in undirected graphs described in [11] is presented: the algorithm, due to Holm, de Lichtenberg and Thorup, answers connectivity queries in $O(\log n/\log\log n)$ worst-case running time while supporting edge insertions and deletions in $O(\log^2 n)$ amortized time.

The algorithm maintains a spanning forest F of the dynamically changing graph G. Edges in F are referred to as *tree edges*. Let e be a tree edge of forest F, and let T be the tree of F containing it. When e is deleted, the two trees T_1 and T_2 obtained from T after the deletion of e can be reconnected if and only if there is a non-tree edge in G with one endpoint in T_1 and the other endpoint in T_2. Such an edge is called a *replacement edge* for e. In other words, if there is a replacement edge for e, T is reconnected via this replacement edge; otherwise, the deletion of e creates a new connected component in G.

To accommodate systematic search for replacement edges, the algorithm associates to each edge e a level $\ell(e)$ and, based on edge levels, maintains a set of sub-forests of the spanning forest F: for each level i, forest F_i is the sub-forest induced by tree edges of level $\geq i$. Denoting by L denotes the maximum edge level, it follows that:

$$F = F_0 \supseteq F_1 \supseteq F_2 \supseteq \cdots \supseteq F_L .$$

Initially, all edges have level 0; levels are then progressively increased, but never decreased. The changes of edge levels are accomplished so as to maintain the following invariants, which obviously hold at the beginning.

Invariant (1): F is a maximum spanning forest of G if edge levels are interpreted as weights.

Invariant (2): The number of nodes in each tree of F_i is at most $n/2^i$.

Invariant (1) should be interpreted as follows. Let (u, v) be a non-tree edge of level $\ell(u, v)$ and let $u \cdots v$ be the unique path between u and v in F (such a path exists since F is a spanning forest of G). Let e be any edge in $u \cdots v$ and let $\ell(e)$ be its level. Due to (1), $\ell(e) \geq \ell(u, v)$. Since this holds for each edge in the path, and by construction $F_{\ell(u,v)}$ contains all the tree edges of level $\geq \ell(u, v)$, the entire path is contained in $F_{\ell(u,v)}$, i. e., u and v are connected in $F_{\ell(u,v)}$.

Invariant (2) implies that the maximum number of levels is $L \leq \lfloor \log_2 n \rfloor$.

Note that when a new edge is inserted, it is given level 0. Its level can be then increased at most $\lfloor \log_2 n \rfloor$ times as a consequence of edge deletions. When a tree edge $e = (v, w)$ of level $\ell(e)$ is deleted, the algorithm looks for a replacement edge at the highest possible level, if any. Due to invariant (1), such a replacement edge has level $\ell \leq \ell(e)$. Hence, a replacement subroutine `Replace`$((u, w), \ell(e))$ is called with parameters e and $\ell(e)$. The operations performed by this subroutine are now sketched.

`Replace`$((u, w), \ell)$ finds a replacement edge of the highest level $\leq \ell$, if any. If such a replacement does not exist in level ℓ, there are two cases: if $\ell > 0$, the algorithm recurses on level $\ell - 1$; otherwise, $\ell = 0$, and the deletion of (v, w) disconnects v and w in G.

During the search at level ℓ, suitably chosen tree and non-tree edges may be promoted at higher levels as follows. Let T_v and T_w be the trees of forest F_ℓ obtained after deleting (v, w) and let, w.l.o.g., T_v be smaller than T_w. Then T_v contains at most $n/2^{\ell+1}$ vertices, since $T_v \cup T_w \cup \{(v, w)\}$ was a tree at level ℓ and due to invariant (2). Thus, edges in T_v of level ℓ can be promoted at level $\ell + 1$ by maintaining the invariants. Non-tree edges incident to T_v are finally visited one by one: if an edge does connect T_v and T_w, a replacement edge has been found and the search stops, otherwise its level is increased by 1.

Trees of each forest are maintained so that the basic operations needed to implement edge insertions and deletions can be supported in $O(\log n)$ time. There are few variants of basic data structures that can accomplish this task, and one could use the Euler Tour trees (in short ET-tree), first introduced in [17], for this purpose.

In addition to inserting and deleting edges from a forest, ET-trees must also support operations such as finding the tree of a forest that contains a given vertex, computing the size of a tree, and, more importantly, finding tree edges of level ℓ in T_v and non-tree edges of level ℓ incident to T_v. This can be done by augmenting the ET-trees with a constant amount of information per node: the interested reader is referred to [11] for details.

Using an amortization argument based on level changes, the claimed $O(\log^2 n)$ bound on the update time can be proved. Namely, inserting an edge costs $O(\log n)$, as well as increasing its level. Since this can happen $O(\log n)$ times, the total amortized insertion cost, inclusive of level increases, is $O(\log^2 n)$. With respect to edge deletions, cutting and linking $O(\log n)$ forest has a total cost $O(\log^2 n)$; moreover, there are $O(\log n)$ recursive calls to `Replace`, each of cost $O(\log n)$ plus the cost amortized over level increases. The ET-trees over $F_0 = F$ allows it to answer connectivity queries in $O(\log n)$ worst-case time. As shown in [11], this can be reduced to $O(\log n/\log\log n)$ by using a $\Theta(\log n)$-ary version of ET-trees.

Theorem 1 *A dynamic graph G with n vertices can be maintained upon insertions and deletions of edges using $O(\log^2 n)$ amortized time per update and answering connectivity queries in $O(\log n/\log\log n)$ worst-case running time.*

Later on, Thorup [18] gave another data structure which achieves slightly different time bounds:

Theorem 2 *A dynamic graph G with n vertices can be maintained upon insertions and deletions of edges using $O(\log n \cdot (\log\log n)^3)$ amortized time per update and answering connectivity queries in $O(\log n/\log\log\log n)$ time.*

The bounds given in Theorems 1 and 2 are not directly comparable, because each sacrifices the running time of one operation (either query or update) in order to improve the other.

The best known lower bound for the dynamic connectivity problem holds in the bit-probe model of computation and is due to Pătraşcu and Tarniţă [16]. The bit-probe model is an instantiation of the cell-probe model with one-bit cells. In this model, memory is organized in cells, and the algorithms may read or write a cell in constant time. The number of cell probes is taken as the measure of complexity. For formal definitions of this model, the interested reader is referred to [13].

Theorem 3 *Consider a bit-probe implementation for dynamic connectivity, in which updates take expected amortized time t_u, and queries take expected time t_q. Then, in the average case of an input distribution, $t_u = \Omega\left(\log^2 n/\log^2(t_u + t_q)\right)$. In particular*

$$\max\{t_u, t_q\} = \Omega\left(\left(\frac{\log n}{\log\log n}\right)^2\right).$$

In the bit-probe model, the best upper bound per operation is given by the algorithm of Theorem 2, namely it is $O(\log^2 n/\log\log\log n)$. Consequently, the gap between upper and lower bound appears to be limited essentially to doubly logarithmic factors only.

Applications

Dynamic graph connectivity appears as a basic subproblem of many other important problems, such as the dynamic maintenance of minimum spanning trees and dynamic edge and vertex connectivity problems. Furthermore, there are several applications of dynamic graph connectivity in other disciplines, ranging from Computational Biology, where dynamic graph connectivity proved to be useful for the dynamic maintenance of protein molecular surfaces as the molecules undergo conformational changes [6], to Image Processing, when one is interested in maintaining the connected components of a bitmap image [3].

Open Problems

The work on dynamic connectivity raises some open and perhaps intruiguing questions. The first natural open problem is whether the gap between upper and lower bounds can be closed. Note that the lower bound of Theorem 3 seems to imply that different trade-offs between queries and updates could be possible: can we design a data structure with $o(\log n)$ time per update and $O(\text{poly}(\log n))$ per query? This would be particulary interesting in applications where the total number of queries is substantially larger than the number of updates.

Finally, is it possible to design an algorithm with matching $O(\log n)$ update and query bounds for general graphs? Note that this is possible in the special case of plane graphs [5].

Experimental Results

A thorough empirical study of dynamic connectivity algorithms has been carried out in [1,12].

Data Sets

Data sets are described in [1,12].

Cross References

► Dynamic Trees
► Fully Dynamic All Pairs Shortest Paths
► Fully Dynamic Higher Connectivity
► Fully Dynamic Higher Connectivity for Planar Graphs

▶ Fully Dynamic Minimum Spanning Trees
▶ Fully Dynamic Planarity Testing
▶ Fully Dynamic Transitive Closure

Recommended Reading

1. Alberts, D., Cattaneo, G., Italiano, G.F.: An empirical study of dynamic graph algorithms. ACM J. Exp. Algorithmics **2** (1997)
2. Beame, P., Fich, F.E.: Optimal bounds for the predecessor problem and related problems. J. Comp. Syst. Sci. **65**(1), 38–72 (2002)
3. Eppstein, D.: Dynamic Connectivity in Digital Images. Inf. Process. Lett. **62**(3), 121–126 (1997)
4. Eppstein, D., Galil, Z., Italiano, G.F., Nissenzweig, A.: Sparsification – a technique for speeding up dynamic graph algorithms. J. Assoc. Comp. Mach. **44**(5), 669–696 (1997)
5. Eppstein, D., Italiano, G.F., Tamassia, R., Tarjan, R.E., Westbrook, J., Yung, M.: Maintenance of a minimum spanning forest in a dynamic plane graph. J. Algorithms **13**, 33–54 (1992)
6. Eyal, E., Halperin, D.: Improved Maintenance of Molecular Surfaces Using Dynamic Graph Connectivity. in: Proc. 5th International Workshop on Algorithms in Bioinformatics (WABI 2005), Mallorca, Spain, 2005, pp. 401–413
7. Frederickson, G.N.: Data structures for on-line updating of minimum spanning trees. SIAM J. Comp. **14**, 781–798 (1985)
8. Frederickson, G.N.: Ambivalent data structures for dynamic 2-edge-connectivity and k smallest spanning trees. In: Proc. 32nd Symp. Foundations of Computer Science, 1991, pp. 632–641
9. Henzinger, M.R., Fredman, M.L.: Lower bounds for fully dynamic connectivity problems in graphs. Algorithmica **22**(3), 351–362 (1998)
10. Henzinger, M.R., King, V.: Randomized fully dynamic graph algorithms with polylogarithmic time per operation. J. ACM **46**(4), 502–516 (1999)
11. Holm, J., de Lichtenberg, K., Thorup, M.: Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. J. ACM **48**, 723–760 (2001)
12. Iyer, R., Karger, D., Rahul, H., Thorup, M.: An Experimental Study of Polylogarithmic, Fully Dynamic, Connectivity Algorithms. ACM J. Exp. Algorithmics **6** (2001)
13. Miltersen, P.B.: Cell probe complexity – a survey. In: 19th Conference on the Foundations of Software Technology and Theoretical Computer Science (FSTTCS), Advances in Data Structures Workshop, 1999
14. Miltersen, P.B., Subramanian, S., Vitter, J.S., Tamassia, R.: Complexity models for incremental computation. In: Ausiello, G., Italiano, G.F. (eds.) Special Issue on Dynamic and On-line Algorithms. Theor. Comp. Sci. **130**(1), 203–236 (1994)
15. Pǎtraşcu, M., Demain, E.D.: Lower Bounds for Dynamic Connectivity. In: Proc. 36th ACM Symposium on Theory of Computing (STOC), 2004, pp. 546–553
16. Pǎtraşcu, M., Tarniţǎ, C.: On Dynamic Bit-Probe Complexity, Theoretical Computer Science, Special Issue on ICALP'05. In: Italiano, G.F., Palamidessi, C. (eds.) vol. 380, pp. 127–142 (2007) A preliminary version in Proc. 32nd International Colloquium on Automata, Languages and Programming (ICALP'05), 2005, pp. 969–981
17. Tarjan, R.E., Vishkin, U.: An efficient parallel biconnectivity algorithm. SIAM J. Comp. **14**, 862–874 (1985)
18. Thorup, M.: Near-optimal fully-dynamic graph connectivity. In: Proc. 32nd ACM Symposium on Theory of Computing (STOC), 2000, pp. 343–350

Fully Dynamic Higher Connectivity

1997; Eppstein, Galil, Italiano, Nissenzweig

GIUSEPPE F. ITALIANO
Department of Information and Computer Systems, University of Rome, Rome, Italy

Keywords and Synonyms

Fully dynamic edge connectivity; Fully dynamic vertex connectivity

Problem Definition

The problem is concerned with efficiently maintaining information about edge and vertex connectivity in a dynamically changing graph. Before defining formally the problems, a few preliminary definitions follow.

Given an undirected graph $G = (V, E)$, and an integer $k \geq 2$, a pair of vertices $\langle u, v \rangle$ is said to be *k-edge-connected* if the removal of any $(k-1)$ edges in G leaves u and v connected. It is not difficult to see that this is an equivalence relationship: the vertices of a graph G are partitioned by this relationship into equivalence classes called *k-edge-connected components*. G is said to be *k-edge-connected* if the removal of any $(k-1)$ edges leaves G connected. As a result of these definitions, G is k-edge-connected if and only if any two vertices of G are k-edge-connected. An edge set $E' \subseteq E$ is an *edge-cut for vertices x and y* if the removal of all the edges in E' disconnects G into two graphs, one containing x and the other containing y. An edge set $E' \subseteq E$ is an *edge-cut for G* if the removal of all the edges in E' disconnects G into two graphs. An edge-cut E' for G (for x and y, respectively) is *minimal* if removing any edge from E' reconnects G (for x and y, respectively). The cardinality of an edge-cut E', denoted by $|E'|$, is given by the number of edges in E'. An edge-cut E' for G (for x and y, respectively) is said to be a *minimum cardinality edge-cut* or in short a *connectivity edge-cut* if there is no other edge-cut E'' for G (for x and y respectively) such that $|E''| < |E'|$. Connectivity edge-cuts are of course minimal edge-cuts. Note that G is k-edge-connected if and only if a connectivity edge-cut for G contains at least k edges, and vertices x and y are k-edge-connected if and only if a connectivity edge-cut for x and y contains at least k edges. A connectivity edge-cut of cardinality 1 is called a *bridge*.

The following theorem due to Ford and Fulkerson, and Elias, Feinstein and Shannon (see [7]) gives another characterization of k-edge connectivity.

Theorem 1 (Ford and Fulkerson, Elias, Feinstein and Shannon) *Given a graph G and two vertices x and y in G, x and y are k-edge-connected if and only if there are at least k edge-disjoint paths between x and y.*

In a similar fashion, a vertex set $V' \subseteq V - \{x, y\}$ is said to be a *vertex-cut* for vertices x and y if the removal of all the vertices in V' disconnects x and y. $V' \subset V$ is a *vertex-cut* for vertices G if the removal of all the vertices in V' disconnects G.

The cardinality of a vertex-cut V', denoted by $|V'|$, is given by the number of vertices in V'. A vertex-cut V' for x and y is said to be a *minimum cardinality vertex-cut* or in short a *connectivity vertex-cut* if there is no other vertex-cut V'' for x and y such that $|V''| < |V'|$. Then x and y are k-vertex-connected if and only if a connectivity vertex-cut for x and y contains at least k vertices. A graph G is said to be *k-vertex-connected* if all its pairs of vertices are k-vertex-connected. A connectivity vertex-cut of cardinality 1 is called an *articulation point*, while a connectivity vertex-cut of cardinality 2 is called a *separation pair*. Note that for vertex connectivity it is no longer true that the removal of a connectivity vertex-cut splits G into two sets of vertices.

The following theorem due to Menger (see [7]) gives another characterization of k-vertex connectivity.

Theorem (Menger) 2 *Given a graph G and two vertices x and y in G, x and y are k-vertex-connected if and only if there are at least k vertex-disjoint paths between x and y.*

A dynamic graph algorithm maintains a given property $\mathcal{P}$ on a graph subject to dynamic changes, such as edge insertions, edge deletions and edge weight updates. A dynamic graph algorithm should process queries on property $\mathcal{P}$ quickly, and perform update operations faster than recomputing from scratch, as carried out by the fastest static algorithm. An algorithm is *fully dynamic* if it can handle both edge insertions and edge deletions. A *partially dynamic* algorithm can handle either edge insertions or edge deletions, but not both: it is *incremental* if it supports insertions only, and *decremental* if it supports deletions only.

In the *fully dynamic k-edge connectivity problem* one wishes to maintain an undirected graph $G = (V, E)$ under an intermixed sequence of the following operations:

- *k-EdgeConnected(u, v):* Return *true* if vertices u and v are in the same k-edge-connected component. Return *false* otherwise.
- *Insert(x, y):* Insert a new edge between the two vertices x and y.
- *Delete(x, y):* Delete the edge between the two vertices x and y.

In the *fully dynamic k-vertex connectivity problem* one wishes to maintain an undirected graph $G = (V, E)$ under an intermixed sequence of the following operations:

- *k-VertexConnected(u, v):* Return *true* if vertices u and v are k-vertex-connected. Return *false* otherwise.
- *Insert(x, y):* Insert a new edge between the two vertices x and y.
- *Delete(x, y):* Delete the edge between the two vertices x and y.

Key Results

To the best knowledge of the author, the most efficient fully dynamic algorithms for k-edge and k-vertex connectivity were proposed in [3,12]. Their running times are characterized by the following theorems.

Theorem 3 *The fully dynamic k-edge connectivity problem can be solved in:*

1. *$O(\log^4 n)$ time per update and $O(\log^3 n)$ time per query, for $k = 2$*
2. *$O(n^{2/3})$ time per update and query, for $k = 3$*
3. *$O(n\alpha(n))$ time per update and query, for $k = 4$*
4. *$O(n \log n)$ time per update and query, for $k \geq 5$.*

Theorem 4 *The fully dynamic k-vertex connectivity problem can be solved in:*

1. *$O(\log^4 n)$ time per update and $O(\log^3 n)$ time per query, for $k = 2$*
2. *$O(n)$ time per update and query, for $k = 3$*
3. *$O(n\alpha(n))$ time per update and query, for $k = 4$.*

Applications

Vertex and edge connectivity problems arise often in issues related to network reliability and survivability. In computer networks, the vertex connectivity of the underlying graph is related to the smallest number of nodes that might fail before disconnecting the whole network. Similarly, the edge connectivity is related to the smallest number of links that might fail before disconnecting the entire network. Analogously, if two nodes are k-vertex-connected then they can remain connected even after the failure of up to $(k - 1)$ other nodes, and if they are k-edge-connected then they can survive the failure of up to $(k - 1)$ links. It is important to investigate the dynamic versions of those problems in contexts where the networks are dynamically evolving, say, when links may go up and down because of failures and repairs.

Open Problems

The work of Eppstein et al. [3] and Holm et al. [12] raises some intriguing questions. First, while efficient dynamic algorithms for k-edge connectivity are known for general k, no efficient fully dynamic k-vertex connectivity is known for $k \geq 5$. To the best of the author's knowledge, in this case even no static algorithm is known. Second, fully dynamic 2-edge and 2-vertex connectivity can be solved in polylogarithmic time per update, while the best known update bounds for higher edge and vertex connectivity are polynomial: Can this gap be reduced, i. e., can one design polylogarithnmic algorithms for fully dynamic 3-edge and 3-vertex connectivity?

Cross References

► Dynamic Trees
► Fully Dynamic All Pairs Shortest Paths
► Fully Dynamic Connectivity
► Fully Dynamic Higher Connectivity for Planar Graphs
► Fully Dynamic Minimum Spanning Trees
► Fully Dynamic Planarity Testing
► Fully Dynamic Transitive Closure

Recommended Reading

1. Dinitz, E.A.: Maintaining the 4-edge-connected components of a graph on-line. In: Proc. 2nd Israel Symp. Theory of Computing and Systems, 1993, pp. 88–99
2. Dinitz, E.A., Karzanov A.V., Lomonosov M.V.: On the structure of the system of minimal edge cuts in a graph. In: Fridman, A.A. (ed) Studies in Discrete Optimization, pp. 290–306. Nauka, Moscow (1990). In Russian
3. Eppstein, D., Galil Z., Italiano G.F., Nissenzweig A.: Sparsification – a technique for speeding up dynamic graph algorithms. J. Assoc. Comput. Mach. **44**(5), 669–696 (1997)
4. Frederickson, G.N.: Ambivalent data structures for dynamic 2-edge-connectivity and k smallest spanning trees. SIAM J. Comput. **26**(2), 484–538 (1997)
5. Galil, Z., Italiano, G. F.: Fully dynamic algorithms for 2-edge-connectivity. SIAM J. Comput. **21**, 1047–1069 (1992)
6. Galil, Z., Italiano, G.F.: Maintaining the 3-edge-connected components of a graph on-line. SIAM J. Comput. **22**, 11–28 (1993)
7. Harary, F.: Graph Theory. Addison-Wesley, Reading (1969)
8. Henzinger, M.R.: Fully dynamic biconnectivity in graphs. Algorithmica **13**(6), 503–538 (1995)
9. Henzinger, M.R.: Improved data structures for fully dynamic biconnectivity. SIAM J. Comput. **29**(6), 1761–1815 (2000)
10. Henzinger, M., King V.: Fully dynamic biconnectivity and transitive closure. In: Proc. 36th IEEE Symposium on Foundations of Computer Science (FOCS'95), 1995, pp. 664–672
11. Henzinger, M.R., King, V.: Randomized fully dynamic graph algorithms with polylogarithmic time per operation. J. ACM **46**(4), 502–516 (1999)
12. Holm, J., de Lichtenberg, K., Thorup, M.: Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. J. ACM **48**, 723–760 (2001)
13. Karzanov, A.V., Timofeev, E. A.: Efficient algorithm for finding all minimal edge cuts of a nonoriented graph. Cybernetics **22**, 156–162 (1986)
14. La Poutré, J.A.: Maintenance of triconnected components of graphs. In: Proc. 19th Int. Colloquium on Automata, Languages and Programming. Lecture Notes in Computer Science, vol. 623, pp. 354–365. Springer, Berlin (1992)
15. La Poutré, J.A.: Maintenance of 2- and 3-edge-connected components of graphs II. SIAM J. Comput. **29**(5), 1521–1549 (2000)
16. La Poutré, J.A., van Leeuwen, J., Overmars, M.H.: Maintenance of 2- and 3-connected components of graphs, part I: 2- and 3-edge-connected components. Discret. Math. **114**, 329–359 (1993)
17. La Poutré, J.A., Westbrook, J.: Dynamic two-connectivity with backtracking. In: Proc. 5th ACM-SIAM Symp. Discrete Algorithms, 1994, pp. 204–212
18. Westbrook, J., Tarjan, R.E.: Maintaining bridge-connected and biconnected components on-line. Algorithmica **7**, 433–464 (1992)

Fully Dynamic Higher Connectivity for Planar Graphs

1998; Eppstein, Galil, Italiano, Spencer

GIUSEPPE F. ITALIANO
Department of Information and Computer Systems, University of Rome, Rome, Italy

Keywords and Synonyms

Fully dynamic edge connectivity; Fully dynamic vertex connectivity

Problem Definition

In this entry, the problem of maintaining a dynamic planar graph subject to edge insertions and edge deletions that preserve planarity but that can change the embedding is considered. In particular, in this problem one is concerned with the problem of efficiently maintaining information about edge and vertex connectivity in such a dynamically changing planar graph. The algorithms to solve this problem must handle insertions that keep the graph planar without regard to any particular embedding of the graph. The interested reader is referred to the chapter "Fully Dynamic Planarity Testing" of this encyclopedia for algorithms to learn how to check efficiently whether a graph subject to edge insertions and deletions remains planar (without regard to any particular embedding).

Before defining formally the problems considered here, a few preliminary definitions follow.

Given an undirected graph $G = (V, E)$, and an integer $k \geq 2$, a pair of vertices $\langle u, v\rangle$ is said to be *k-edge-connected* if the removal of any $(k-1)$ edges in G leaves u and v connected. It is not difficult to see that this is an equivalence relationship: the vertices of a graph G are partitioned by this relationship into equivalence classes called *k-edge-connected components*. G is said to be *k-edge-connected* if the removal of any $(k-1)$ edges leaves G connected. As a result of these definitions, G is k-edge-connected if and only if any two vertices of G are k-edge-connected. An edge set $E' \subseteq E$ is an *edge-cut for vertices x and y* if the removal of all the edges in E' disconnects G into two graphs, one containing x and the other containing y. An edge set $E' \subseteq E$ is an *edge-cut for G* if the removal of all the edges in E' disconnects G into two graphs. An edge-cut E' for G (for x and y, respectively) is *minimal* if removing any edge from E' reconnects G (for x and y, respectively). The cardinality of an edge-cut E', denoted by $|E'|$, is given by the number of edges in E'. An edge-cut E' for G (for x and y, respectively) is said to be a *minimum cardinality edge-cut* or in short a *connectivity edge-cut* if there is no other edge-cut E'' for G (for x and y, respectively) such that $|E''| < |E'|$. Connectivity edge-cuts are of course minimal edge-cuts. Note that G is k-edge-connected if and only if a connectivity edge-cut for G contains at least k edges, and vertices x and y are k-edge-connected if and only if a connectivity edge-cut for x and y contains at least k edges. A connectivity edge-cut of cardinality 1 is called a *bridge*.

In a similar fashion, a vertex set $V' \subseteq V - \{x, y\}$ is said to be a *vertex-cut* for vertices x and y if the removal of all the vertices in V' disconnects x and y. $V' \subset V$ is a *vertex-cut* for vertices G if the removal of all the vertices in V' disconnects G.

The cardinality of a vertex-cut V', denoted by $|V'|$, is given by the number of vertices in V'. A vertex-cut V' for x and y is said to be a *minimum cardinality vertex-cut* or in short a *connectivity vertex-cut* if there is no other vertex-cut V'' for x and y such that $|V''| < |V'|$. Then x and y are k-vertex-connected if and only if a connectivity vertex-cut for x and y contains at least k vertices. A graph G is said to be *k-vertex-connected* if all its pairs of vertices are k-vertex-connected. A connectivity vertex-cut of cardinality 1 is called an *articulation point*, while a connectivity vertex-cut of cardinality 2 is called a *separation pair*. Note that for vertex connectivity it is no longer true that the removal of a connectivity vertex-cut splits G into two sets of vertices.

A dynamic graph algorithm maintains a given property $\mathcal{P}$ on a graph subject to dynamic changes, such as edge insertions, edge deletions and edge weight updates. A dynamic graph algorithm should process queries on property $\mathcal{P}$ quickly, and perform update operations faster than recomputing from scratch, as carried out by the fastest static algorithm. An algorithm is *fully dynamic* if it can handle both edge insertions and edge deletions. A *partially dynamic* algorithm can handle either edge insertions or edge deletions, but not both: it is *incremental* if it supports insertions only, and *decremental* if it supports deletions only.

In the *fully dynamic k-edge connectivity problem for a planar graph* one wishes to maintain an undirected planar graph $G = (V, E)$ under an intermixed sequence of edge insertions, edge deletions and queries about the k-edge connectivity of the underlying planar graph. Similarly, in the*fully dynamic k-vertex connectivity problem for a planar graph* one wishes to maintain an undirected planar graph $G = (V, E)$ under an intermixed sequence of edge insertions, edge deletions and queries about the k-vertex connectivity of the underlying planar graph.

Key Results

The algorithms in [2,3] solve efficiently the above problems for small values of k:

Theorem 1 *One can maintain a planar graph, subject to insertions and deletions that preserve planarity, and allow queries that test the 2-edge connectivity of the graph, or test whether two vertices belong to the same 2-edge-connected component, in $O(\log n)$ amortized time per insertion or query, and $O(\log^2 n)$ per deletion.*

Theorem 2 *One can maintain a planar graph, subject to insertions and deletions that preserve planarity, and allow testing of the 3-edge and 4-edge connectivity of the graph in $O(n^{1/2})$ time per update, or testing of whether two vertices are 3- or 4-edge-connected, in $O(n^{1/2})$ time per update or query.*

Theorem 3 *One can maintain a planar graph, subject to insertions and deletions that preserve planarity, and allow queries that test the 3-vertex connectivity of the graph, or test whether two vertices belong to the same 3-vertex-connected component, in $O(n^{1/2})$ amortized time per update or query.*

Note that these theorems improve on the bounds known for the same problems on general graphs, reported in the chapter "Fully Dynamic Higher Connectivity."

Applications

The interest reader is referred to the chapter "Fully Dynamic Higher Connectivity" for applications of dynamic edge and vertex connectivity. The case of planar graphs is especially important, as these graphs arise frequently in applications.

Open Problems

A number of problems related to the work of Eppstein et al. [2,3] remain open. First, can the running times per operation be improved? Second, as in the case of general graphs, also for planar graphs fully dynamic 2-edge connectivity can be solved in polylogarithmic time per update, while the best known update bounds for higher edge and vertex connectivity are polynomial: Can this gap be reduced, i. e., can one design polylogarithnmic algorithms at least for fully dynamic 3-edge and 3-vertex connectivity? Third, in the special case of planar graphs can one solve fully dynamic k-vertex connectivity for general k?

Cross References

▶ Dynamic Trees
▶ Fully Dynamic All Pairs Shortest Paths
▶ Fully Dynamic Connectivity
▶ Fully Dynamic Higher Connectivity
▶ Fully Dynamic Minimum Spanning Trees
▶ Fully Dynamic Planarity Testing
▶ Fully Dynamic Transitive Closure

Recommended Reading

1. Galil Z., Italiano G.F., Sarnak N.: Fully dynamic planarity testing with applications. J. ACM **48**, 28–91 (1999)
2. Eppstein D., Galil Z., Italiano G.F., Spencer T.H.: Separator based sparsification I: planarity testing and minimum spanning trees. J. Comput. Syst. Sci., Special issue of STOC 93 **52**(1), 3–27 (1996)
3. Eppstein D., Galil Z., Italiano G.F., Spencer T.H.: Separator based sparsification II: edge and vertex connectivity. SIAM J. Comput. **28**, 341–381 (1999)
4. Giammarresi D., Italiano G.F.: Decremental 2- and 3-connectivity on planar graphs. Algorithmica **16**(3), 263–287 (1996)
5. Hershberger J., M.R., Suri S.: Data structures for two-edge connectivity in planar graphs. Theor. Comput. Sci. **130**(1), 139–161 (1994)

Fully Dynamic Minimum Spanning Trees

2000; Holm, de Lichtenberg, Thorup

GIUSEPPE F. ITALIANO
Department of Information and Computer Systems, University of Rome, Rome, Italy

Keywords and Synonyms

Dynamic minimum spanning forests

Problem Definition

Let $G = (V, E)$ be an undirected weighted graph. The problem considered here is concerned with maintaining efficiently information about a minimum spanning tree of G (or minimum spanning forest if G is not connected), when G is subject to dynamic changes, such as edge insertions, edge deletions and edge weight updates. One expects from the dynamic algorithm to perform update operations faster than recomputing the entire minimum spanning tree from scratch.

Throughout, an algorithm is said to be *fully dynamic* if it can handle both edge insertions and edge deletions. A *partially dynamic* algorithm can handle either edge insertions or edge deletions, but not both: it is *incremental* if it supports insertions only, and *decremental* if it supports deletions only.

Key Results

The dynamic minimum spanning forest algorithm presented in this section builds upon the dynamic connectivity algorithm described in the entry "Fully Dynamic Connectivity". In particular, a few simple changes to that algorithm are sufficient to maintain a minimum spanning forest of a weighted undirected graph upon deletions of edges [13]. A general reduction from [11] can then be applied to make the deletions-only algorithm fully dynamic.

This section starts by describing a decremental algorithm for maintaining a minimum spanning forest under deletions only. Throughout the sequence of deletions, the algorithm maintains a minimum spanning forest F of the dynamically changing graph G. The edges in F are referred to as *tree edges* and the other edges (in $G - F$) are referred to as *non-tree edges*. Let e be an edge being deleted. If e is a non-tree edge, then the minimum spanning forest does not need to change, so the interesting case is when e is a tree edge of forest F. Let T be the tree of F containing e. In this case, the deletion of e disconnects the tree T into two trees T_1 and T_2: to update the minimum spanning forest, one has to look for the minimum weight edge having one endpoint in T_1 and the other endpoint in T_2. Such an edge is called a *replacement edge* for e.

As for the dynamic connectivity algorithm, to search for replacement edges, the algorithm associates to each edge e a level $\ell(e)$ and, based on edge levels, maintains a set of sub-forests of the minimum spanning forest F: for each level i, forest F_i is the sub-forest induced by tree edges of level $\geq i$. Denoting by L the maximum edge level, it follows that:

$$F = F_0 \supseteq F_1 \supseteq F_2 \supseteq \cdots \supseteq F_L \, .$$

Initially, all edges have level 0; levels are then progressively increased, but never decreased. The changes of edge levels are accomplished so as to maintain the following invariants, which obviously hold at the beginning.

Invariant (1): F is a maximum spanning forest of G if edge levels are interpreted as weights.

Invariant (2): The number of nodes in each tree of F_i is at most $n/2^i$.

Invariant (3): Every cycle C has a non-tree edge of maximum weight and minimum level among all the edges in C.

Invariant (1) should be interpreted as follows. Let (u,v) be a non-tree edge of level $\ell(u, v)$ and let $u \cdots v$ be the unique path between u and v in F (such a path exists since F is a spanning forest of G). Let e be any edge in $u \cdots v$ and let $\ell(e)$ be its level. Due to (1), $\ell(e) \geq \ell(u, v)$. Since this holds for each edge in the path, and by construction $F_{\ell(u,v)}$ contains all the tree edges of level $\geq \ell(u, v)$, the entire path is contained in $F_{\ell(u,v)}$, i. e., u and v are connected in $F_{\ell(u,v)}$.

Invariant (2) implies that the maximum number of levels is $L \leq \lfloor \log_2 n \rfloor$.

Invariant (3) can be used to prove that, among all the replacement edges, the lightest edge is on the maximum level. Let e_1 and e_2 be two replacement edges with $w(e_1) < w(e_2)$, and let C_i be the cycle induced by e_i in F, $i = 1, 2$. Since F is a minimum spanning forest, e_i has maximum weight among all the edges in C_i. In particular, since by hypothesis $w(e_1) < w(e_2)$, e_2 is also the heaviest edge in cycle $C = (C_1 \cup C_2) \setminus (C_1 \cap C_2)$. Thanks to Invariant (3), e_2 has minimum level in C, proving that $\ell(e_2) \leq \ell(e_1)$. Thus, considering non-tree edges from higher to lower levels is correct.

Note that initially, an edge is is given level 0. Its level can be then increased at most $\lfloor \log_2 n \rfloor$ times as a consequence of edge deletions. When a tree edge $e = (v, w)$ of level $\ell(e)$ is deleted, the algorithm looks for a replacement edge at the highest possible level, if any. Due to invariant (1), such a replacement edge has level $\ell \leq \ell(e)$. Hence, a replacement subroutine `Replace`$((u, w), \ell(e))$ is called with parameters e and $\ell(e)$. The operations performed by this subroutine are now sketched.

`Replace`$((u, w), \ell)$ finds a replacement edge of the highest level $\leq \ell$, if any, considering edges in order of increasing weight. If such a replacement does not exist in level ℓ, there are two cases: if $\ell > 0$, the algorithm recurses on level $\ell - 1$; otherwise, $\ell = 0$, and the deletion of (v,w) disconnects v and w in G.

It is possible to show that `Replace` returns a replacement edge of minimum weight on the highest possible level, yielding the following lemma:

Lemma 1 *There exists a deletions-only minimum spanning forest algorithm that can be initialized on a graph with n vertices and m edges and supports any sequence of edge deletions in $O(m \log^2 n)$ total time.*

The description of a fully dynamic algorithm which performs updates in $O(\log^4 n)$ time now follows. The reduction used to obtain a fully dynamic algorithm is a slight generalization of the construction proposed by Henzinger and King [11] and works as follows.

Lemma 2 *Suppose there is a deletions-only minimum spanning tree algorithm that, for any k and ℓ, can be initialized on a graph with k vertices and ℓ edges and supports any sequence of $\Omega(\ell)$ deletions in total time $O(\ell \cdot t(k, \ell))$, where t is a non-decreasing function. Then there exists a fully-dynamic minimum spanning tree algorithm for a graph with n nodes starting with no edges, that, for m edges, supports updates in time*

$$O\left(\log^3 n + \sum_{i=1}^{3+\log_2 m} \sum_{j=1}^{i} t\left(min\{n, 2^j\}, 2^j\right)\right) .$$

The interested reader is referred to references [11] and [13] for the description of the construction that proves Lemma 2. From Lemma 1 one gets $t(k, \ell) = O(\log^2 k)$. Hence, combining Lemmas 1 and 2, the claimed result follows:

Theorem 3 *There exists a fully-dynamic minimum spanning forest algorithm that, for a graph with n vertices, starting with no edges, maintains a minimum spanning forest in $O(\log^4 n)$ amortized time per edge insertion or deletion.*

There is a lower bound of $\Omega(\log n)$ for dynamic minimum spanning tree, given by *Eppstein et al.* [6], which uses the following argument. Let A be an algorithm for maintaining a minimum spanning tree of an arbitrary (multi)graph G. Let A be such that change weight(e, Δ) returns the edge f that replace e in the minimum spanning tree, if e is replaced. Clearly, any dynamic spanning tree algorithm can be modified to return f. One can use algorithm A to sort n positive numbers $x_1, x_2, \ldots, x_n$, as follows. Construct a multigraph G consisting of two nodes connected by $(n + 1)$ edges $e_0, e_1, \ldots, e_n$, such that edge e_0 has weight 0 and edge e_i has weight x_i. The initial spanning tree is e_0. Increase the weight of e_0 to $+\infty$. Whichever edge replaces e_0, say e_i, is the edge of minimum weight. Now increase the weight of e_i to $+\infty$: the replacement of e_i gives the second smallest weight. Continuing in this fashion gives the numbers sorted in increasing order. A similar argument applies when only edge decreases are allowed. Since Paul and Simon [14] have shown that any sorting algorithm needs $\Omega(n \log n)$ time to sort n numbers on a unit-cost random access machine whose repertoire of operations include additions, subtractions, multiplications and comparisons with 0, but not divisions or bit-wise Boolean operations, the following theorem follows.

Theorem 4 *Any unit-cost random access algorithm that performs additions, subtractions, multiplications and comparisons with* 0*, but not divisions or bit-wise Boolean operations, requires* $\Omega(\log n)$ *amortized time per operation to maintain a minimum spanning tree dynamically.*

Applications

Minimum spanning trees have applications in many areas, including network design, VLSI, and geometric optimization, and the problem of maintaining minimum spanning trees dynamically arises in such applications.

Algorithms for maintaining a minimum spanning forest of a graph can be used also for maintaining information about the connected components of a graph. There are also other applications of dynamic minimum spanning trees algorithms, which include finding the k smallest spanning trees [3,4,5,8,9], sampling spanning trees [7] and dynamic matroid intersection problems [10]. Note that the first two problems are not necessarily dynamic: however, efficient solutions for these problems need dynamic data structures.

Open Problems

The first natural open question is to ask whether the gap between upper and lower bounds for the dynamic minimum spanning tree problem can be closed. Note that this is possible in the special case of plane graphs [6].

Second, the techniques for dynamic minimum spanning trees can be extended to dynamic 2-edge and 2-vertex connectivity, which indeed can be solved in polylogarithmic time per update. Can one extend the same technique also to higher forms of connectivity? This is particularly important, since the best known update bounds for higher edge and vertex connectivity are polynomial, and it would be useful to design polylogarithnmic algorithms at least for fully dynamic 3-edge and 3-vertex connectivity.

Experimental Results

A thorough empirical study on the performance evaluation of dynamic minimum spanning trees algorithms has been carried out in [1,2].

Data Sets

Data sets are described in [1,2].

Cross References

▶ Dynamic Trees
▶ Fully Dynamic All Pairs Shortest Paths
▶ Fully Dynamic Connectivity
▶ Fully Dynamic Higher Connectivity
▶ Fully Dynamic Higher Connectivity for Planar Graphs
▶ Fully Dynamic Planarity Testing
▶ Fully Dynamic Transitive Closure

Recommended Reading

1. Alberts, D., Cattaneo, G., Italiano, G.F.: An empirical study of dynamic graph algorithms. ACM. J. Exp. Algorithm **2**, (1997)
2. Cattaneo, G., Faruolo, P., Ferraro Petrillo, U., Italiano, G.F.: Maintaining Dynamic Minimum Spanning Trees: An Experimental Study. In: Proceeding 4th Workshop on Algorithm Engineering and Experiments (ALENEX 02), 6–8 Jan 2002. pp. 111–125
3. Eppstein, D.: Finding the k smallest spanning trees. BIT. **32**, 237–248 (1992)
4. Eppstein, D.: Tree-weighted neighbors and geometric k smallest spanning trees. Int. J. Comput. Geom. Appl. **4**, 229–238 (1994)
5. Eppstein, D., Galil, Z., Italiano, G.F., Nissenzweig, A.: Sparsification – a technique for speeding up dynamic graph algorithms. J. Assoc. Comput. Mach. **44**(5), 669–696 (1997)
6. Eppstein, D., Italiano, G.F., Tamassia, R., Tarjan, R.E., Westbrook, J., Yung, M.: Maintenance of a minimum spanning forest in a dynamic plane graph. J. Algorithms **13**, 33–54 (1992)
7. Feder, T., Mihail, M.: Balanced matroids. In: Proceeding 24th ACM Symp. Theory of Computing, pp 26–38, Victoria, British Columbia, Canada, May 04–06 1992
8. Frederickson, G.N.: Data structures for on-line updating of minimum spanning trees. SIAM. J. Comput. **14**, 781–798 (1985)
9. Frederickson, G.N.: Ambivalent data structures for dynamic 2-edge-connectivity and k smallest spanning trees. In: Proceeding 32nd Symp. Foundations of Computer Science, pp 632–641, San Juan, Puerto Rico, October 01–04 1991
10. Frederickson, G.N., Srinivas, M.A.: Algorithms and data structures for an expanded family of matroid intersection problems. SIAM. J. Comput. **18**, 112–138 (1989)
11. Henzinger, M.R., King, V.: Maintaining minimum spanning forests in dynamic graphs. SIAM. J. Comput. **31**(2), 364–374 (2001)
12. Henzinger, M.R., King, V.: Randomized fully dynamic graph algorithms with polylogarithmic time per operation. J. ACM **46**(4), 502–516 (1999)
13. Holm, J., de Lichtenberg, K., Thorup, M.: Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. J. ACM **48**, 723–760 (2001)
14. Paul, J., Simon, W.: Decision trees and random access machines. In: Symposium über Logik und Algorithmik. (1980) See also Mehlhorn, K.: Sorting and Searching, pp. 85–97. Springer, Berlin (1984)

15. Tarjan, R.E., Vishkin, U.: An efficient parallel biconnectivity algorithm. SIAM. J. Comput. **14**, 862–874 (1985)

Fully Dynamic Planarity Testing

1999; Galil, Italiano, Sarnak

GIUSEPPE F. ITALIANO
Department of Information and Computer Systems, University of Rome, Rome, Italy

Problem Definition

In this entry, the problem of maintaining a dynamic planar graph subject to edge insertions and edge deletions that preserve planarity but that can change the embedding is considered. Before formally defining the problem, few preliminary definitions follow.

A graph is *planar* if it can be embedded in the plane so that no two edges intersect. In a dynamic framework, a planar graph that is committed to an embedding is called *plane*, and the general term *planar* is used only when changes in the embedding are allowed. An edge insertion that preserves the embedding is called *embedding-preserving*, whereas it is called *planarity-preserving* if it keeps the graph planar, even though its embedding can change; finally, an edge insertion is called *arbitrary* if it is not known to preserve planarity. Extensive work on dynamic graph algorithms has used ad hoc techniques to solve a number of problems such as minimum spanning forests, 2-edge-connectivity and planarity testing for plane graphs (with embedding-preserving insertions) [5,6,7,9,10,11,12]: this entry is concerned with more general planarity-preserving updates.

The work of Galil et al. [8] and of Eppstein et al. [3] provides a general technique for dynamic planar graph problems, including those mentioned above: in all these problems, one can deal with either arbitrary or planarity-preserving insertions and therefore allow changes of the embedding.

The *fully dynamic planarity testing problem* can be defined as follows. One wishes to maintain a (not necessarily planar) graph subject to *arbitrary* edge insertions and deletions, and allow queries that test whether the graph is currently planar, or whether a potential new edge would violate planarity.

Key Results

Eppstein et al. [3] provided a way to apply the sparsification technique [2] to families of graphs that are already sparse, such as planar graphs.

The new ideas behind this technique are the following. The notion of a certificate can be expanded to a definition for graphs in which a subset of the vertices are denoted as *interesting*; these *compressed certificates* may reduce the size of the graph by removing uninteresting vertices. Using this notion, one can define a type of sparsification based on *separators*, small sets of vertices the removal of which splits the graph into roughly equal size components. Recursively finding separators in these components gives a *separator tree* which can also be used as a *sparsification tree*; the interesting vertices in each certificate will be those vertices used in separators at higher levels of the tree. The notion of a *balanced separator tree*, which also partitions the interesting vertices evenly in the tree, is introduced: such a tree can be computed in linear time, and can be maintained dynamically. Using this technique, the following results can be achieved.

Theorem 1 *One can maintain a planar graph, subject to insertions and deletions that preserve planarity, and allow queries that test whether a new edge would violate planarity, in amortized time* $O(n^{1/2})$ *per update or query.*

This result can be improved, in order to allow arbitrary insertions or deletions, even if they might let the graph become nonplanar, using the following approach. The data structure above can be used to maintain a planar subgraph of the given graph. Whenever one attempts to insert a new edge, and the resulting graph would be nonplanar, the algorithm does not actually perform the insertion, but instead adds the edge to a list of *nonplanar edges*. Whenever a query is performed, and the list of nonplanar edges is nonempty, the algorithm attempts once more to add those edges one at a time to the planar subgraph. The time for each successful addition can be charged to the insertion operation that put that edge in the list of nonplanar edges. As soon as the algorithm finds some edge in the list that can not be added, it stops trying to add the other edges in the list. The time for this failed insertion can be charged to the query the algorithm is currently performing. In this way the list of nonplanar edges will be empty if and only if the graph is planar, and the algorithm can test planarity even for updates in nonplanar graphs.

Theorem 2 *One can maintain a graph, subject to arbitrary insertions and deletions, and allow queries that test whether the graph is presently planar or whether a new edge would violate planarity, in amortized time* $O(n^{1/2})$ *per update or query.*

Applications

Planar graphs are perhaps one of the most important interesting subclasses of graphs which combine beautiful structural results with relevance in applications. In particular, planarity testing is a basic problem, which appears naturally in many applications, such as VLSI layout, graphics, and computer aided design. In all these applications, there seems to be a need for dealing with dynamic updates.

Open Problems

The $O(n^{1/2})$ bound for planarity testing is amortized. Can we improve this bound or make it worst-case?

Finally, the complexity of the algorithms presented here, and the large constant factors involved in some of the asymptotic time bounds, make some of the results unsuitable for practical applications. Can one simplify the methods while retaining similar theoretical bounds?

Cross References

- ▶ Dynamic Trees
- ▶ Fully Dynamic All Pairs Shortest Paths
- ▶ Fully Dynamic Connectivity
- ▶ Fully Dynamic Higher Connectivity
- ▶ Fully Dynamic Higher Connectivity for Planar Graphs
- ▶ Fully Dynamic Minimum Spanning Trees
- ▶ Fully Dynamic Transitive Closure

Recommended Reading

1. Cimikowski, R.: Branch-and-bound techniques for the maximum planar subgraph problem. Int. J. Computer Math. **53**, 135–147 (1994)
2. Eppstein, D., Galil, Z., Italiano, G.F., Nissenzweig, A.: Sparsification – a technique for speeding up dynamic graph algorithms. J. Assoc. Comput. Mach. **44**(5), 669–696 (1997)
3. Eppstein, D., Galil, Z., Italiano, G.F., Spencer, T.H.: Separator based sparsification I: planarity testing and minimum spanning trees. J. Comput. Syst. Sci. Special issue of STOC 93 **52**(1), 3–27 (1996)
4. Eppstein, D., Galil, Z., Italiano, G.F., Spencer, T.H.: Separator based sparsification II: edge and vertex connectivity. SIAM J. Comput. **28**, 341–381 (1999)
5. Eppstein, D., Italiano, G.F., Tamassia, R., Tarjan, R.E., Westbrook, J., Yung, M.: Maintenance of a minimum spanning forest in a dynamic plane graph. J. Algorithms **13**, 33–54 (1992)
6. Frederickson, G.N.: Data structures for on-line updating of minimum spanning trees, with applications. SIAM J. Comput. **14**, 781–798 (1985)
7. Frederickson, G.N.: Ambivalent data structures for dynamic 2-edge-connectivity and *k* smallest spanning trees. SIAM J. Comput. **26**(2), 484–538 (1997)
8. Galil, Z., Italiano, G.F., Sarnak, N.: Fully dynamic planarity testing with applications. J. ACM **48**, 28–91 (1999)
9. Giammarresi, D., Italiano, G.F.: Decremental 2- and 3-connectivity on planar graphs. Algorithmica **16**(3):263–287 (1996)
10. Hershberger, J., Suri, M.R., Suri, S.: Data structures for two-edge connectivity in planar graphs. Theor. Comput. Sci. **130**(1), 139–161 (1994)
11. Italiano, G.F., La Poutré, J.A., Rauch, M.: Fully dynamic planarity testing in planar embedded graphs. 1st Annual European Symposium on Algorithms, Bad Honnef, Germany, 30 September–2 October 1993
12. Tamassia, R.: A dynamic data structure for planar graph embedding. 15th Int. Colloq. Automata, Languages, and Programming. LNCS, vol. 317, pp. 576–590. Springer, Berlin (1988)

Fully Dynamic Transitive Closure

1999; King

VALERIE KING
Department of Computer Science Department,
University of Victoria,
Victoria, BC, Canada

Keywords and Synonyms

Incremental algorithms for digraphs; Fully dynamic graph algorithm for maintaining transitive closure; All-pairs dynamic reachability

Problem Definition

Design a data structure for a directed graph with a fixed set of node which can process queries of the form "Is there a path from i to j ?" and updates of the form: "Insert edge (i, j)"; "Delete edge (i, j)". The goal is to minimize update and query times, over the worst case sequence of queries and updates. Algorithms to solve this problem are called "fully dynamic" as opposed to "partially dynamic" since both insertions and deletions are allowed.

Key Results

This work [4] gives the first deterministic fully dynamic graph algorithm for maintaining the transitive closure in a directed graph. It uses $O(n^2 \log n)$ amortized time per update and $O(1)$ worst case query time where n is number of nodes in the graph. The basic technique is extended to give fully dynamic algorithms for approximate and exact all-pairs shortest paths problems.

The basic building block of these algorithms is a method of maintaining all-pairs shortest paths with insertions and deletions for distances up to d. For each vertex v, a single-source shortest path tree of depth d which reach v ("In_v") and another tree of vertices which are reached by v ("Out_v") are maintained during any sequence of deletions. Each insert of a set of edges incident to v results in the rebuilding of In_v and $Out_v I$. For each pair of vertices x, y and each length, a count is kept of the number of v such that there is a path from x in In_v to y in Out_v of that length.

To maintain transitive closure, $\lg n$ levels of these trees are maintained for trees of depth 2, where the edges used to construct a forest on one level depend on the paths in the forest of the previous level.

Space required was reduced from $O(n^3)$ to $O(n^2)$ in [6]. A log n factor was shaved off [7,10]. Other tradeoffs between update and query time are given in [1,7,8,9,10]. A deletions only randomized transitive closure algorithm running in $O(mn)$ time overall is given by [8] where m is the initial number of edges in the graph. A simple monte carlo transitive closure algorithm for acyclic graphs is presented in [5]. Dynamic single source reachability in a digraph is presented in [8,9]. All-pairs shortest paths can be maintained with nearly the same update time [2].

Applications

None

Open Problems

Can reachability from a single source in a directed graph be maintained in $o(mn)$ time over a worst case sequence of m deletions?

Can strongly connected components be maintained in $o(mn)$ time over a worst case sequence of m deletions?

Experimental Results

Experimental results on older techniques can be found in [3].

Cross References

▶ All Pairs Shortest Paths in Sparse Graphs
▶ All Pairs Shortest Paths via Matrix Multiplication
▶ Fully Dynamic All Pairs Shortest Paths
▶ Fully Dynamic Connectivity

Recommended Reading

1. Demestrescu, C., Italiano, G.F.: Trade-offs for fully dynamic transitive closure on DAG's: breaking through the $O(n^2)$ barrier, (presented in FOCS 2000). J. ACM **52**(2), 147–156 (2005)
2. Demestrescu, C., Italiano, G.F.: A new approach to dynamic all pairs shortest paths, (presented in STOC 2003). J. ACM **51**(6), 968–992 (2004)
3. Frigioni, D., Miller, T., Nanni, U., Zaroliagis, C.D.: An experimental study of dynamic algorithms for transitive closure. ACM J Exp. Algorithms **6**(9) (2001)
4. King, V.: Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs. In: Proceedings of the 40th Annual IEEE Symposium on Foundation of Computer Science. ComiIEEE FOCS pp. 81–91. IEEE Computer Society, New York (1999)
5. King, V., Sagert, G.: A fully dynamic algorithm for maintaining the transitive closure, (presented in FOCS 1999). JCCS **65**(1), 150–167 (2002)
6. King, V., Thorup, M.: A space saving trick for dynamic transitive closure and shortest path algorithms. In: Proceedings of the 7th Annual International Conference of Computing and Cominatorics, vol. 2108/2001, pp. 269–277. Lect. Notes Comp. Sci. COCOON Springer, Heidelberg (2001)
7. Roditty, L.: A faster and simpler fully dynamic transitive closure. In: Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms. ACM IEEE SODA, pp. 404–412. ACM, Baltimore (2003)
8. Roditty, L., Zwick, U.: Improved dynamic reachability algorithms for directed graphs. In: Proceedings of the 43rd Annual Symposium on Foundation of Computer Science. IEEE FOCS, pp. 679–688 IEEE Computer Society, Vancouver, Canada (2002)
9. Roditty, L., Zwick, U.: A fully dynamic reachability algorithm for directed graphs with an almost linear update time. In: Proceedings of the 36th ACM Symposium on Theory of Computing. ACM STOC, pp. 184–191 ACM, Chicago (2004)
10. Sankowski, S.: Dynamic transitive closure via dynamic matrix inverse. In: Proceedings of the 45th Annual Symposium on Foundations of Computer Science. IEEE FOCS, 509–517, IEEE Computer Society, Rome, Italy (2004)

G

Gate Sizing

2002; Sundararajan, Sapatnekar, Parhi

VIJAY SUNDARARAJAN
Texas Instruments, Dallas, TX, USA

Keywords and Synonyms

Fast and exact transistor sizing

Problem Definition

For a detailed exposition of the solution approach presented in this article please refer to [15]. As evidenced by the successive announcement of ever faster computer systems in the past decade, increasing the speed of VLSI systems continues to be one of the major requirements for VLSI system designers today. Faster integrated circuits are making possible newer applications that were traditionally considered difficult to implement in hardware. In this scenario of increasing circuit complexity, reduction of circuit delay in integrated circuits is an important design objective. Transistor sizing is one such task that has been employed for speeding up circuits for quite some time now [6]. Given the circuit topology, the delay of a combinational circuit can be controlled by varying the sizes of transistors in the circuit. Here, the size of a transistor is measured in terms of its channel width, since the channel lengths of MOS transistors in a digital circuit are generally uniform. In any case, what really matters is the ratio of channel width to channel length, and if channel lengths are not uniform, this ratio can be considered as the size. In coarse terms, the circuit delay can usually be reduced by increasing the sizes of certain transistors in the circuit from the minimum size. Hence, making the circuit faster usually entails the penalty of increased circuit area relative to a minimum-sized circuit and the area-delay trade-off involved here is the problem of transistor size optimization. A related problem to transistor sizing is called gate sizing, where a logic gate in a circuit is modeled as an equivalent inverter and the sizing optimization is carried out on this modified circuit with equivalent inverters in place of more complex gates. There is, therefore, a reduction in the number of size parameters corresponding to every gate in the circuit. Needless to say, this is an easier problem to solve than the general transistor sizing problem. Note that gate sizing mentioned here is distinct from library specific gate sizing that is a discrete optimization problem targeted to selecting appropriate gate sizes from an underlying cell library. The gate sizing problem targeted here is one of continuous gate sizing where the gate sizes are allowed to vary in a continuous manner between a minimum and a maximum size. There has been a large amount of work done on transistor sizing [1,2,3,5,6,9,10,12,13], that underlines the importance of this optimization technique. Starting from a minimum-sized circuit, TILOS, [6], uses a greedy strategy for transistor sizing by iteratively sizing transistors in the critical path. A sensitivity factor is calculated for every transistor in the critical path to quantify the gain in circuit speed achieved by a unit upsizing of the transistor. The most sensitive transistor is then bumped up in size by a small constant factor to speed up the circuit. This process is repeated iteratively until the timing requirements are met. The technique is extremely simple to implement and has run-time behavior proportional to the size of the circuit. Its chief drawback is that it does not have guaranteed convergence properties and hence is not an exact optimization technique.

Key Results

The solution presented in the article heretofore referred to as MINFLOTRANSIT was a novel way of solving the transistor sizing problem exactly and in an extremely fast manner. Even though the article treats transistor sizing, in the description, the results apply as well to the less general problem of continuous gate sizing as described earlier. The proposed approach has some similarity in form to [2,5,8] which will be subsequently explained, but the similarity in

Gate Sizing, Table 1

Comparison of TILOS and MINFLOTRANSIT on a Sun Ultrasparc 10 workstation for ISCAS85 and MCNC91 benchmarks for 0.13 um technology. The delay specs. are with respect to a minimum-sized circuit. The optimization approach followed here was gate sizing

Circuit	# Gates	Area Saved over TILOS	Delay Specs.	CPU TIME (TILOS)	CPU TIME (OURS)
Adder32	480	$\leq$ 1%	0.5 D_{min}	2.2 s	5 s
Adder256	3840	$\leq$ 1%	0.5 D_{min}	262 s	608 s
Cm163a	65	2.1%	0.55 D_{min}	0.13 s	0.32 s
Cm162a	71	10.4%	0.5 D_{min}	0.23 s	0.96 s
Parity8	89	37%	0.45 D_{min}	0.68 s	2.15 s
Frg1	177	1.9%	0.7 D_{min}	0.55 s	1.49 s
population	518	6.7%	0.4 D_{min}	57 s	179 s
Pmult8	1431	5%	0.5 D_{min}	637 s	1476 s
Alu2	826	2.6%	0.6 D_{min}	28 s	71 s
C432	160	9.4%	0.4 D_{min}	0.5 s	4.8 s
C499	202	7.2%	0.57 D_{min}	1.47 s	11.26 s
C880	383	4%	0.4 D_{min}	2.7 s	8,2 s
C1355	546	9.5%	0.4 D_{min}	29 s	76 s
C1908	880	4.6%	0.4 D_{min}	36 s	84 s
C2670	1193	9.1%	0.4 D_{min}	27 s	69 s
C3540	1669	7.7%	0.4 D_{min}	226 s	651 s
C5315	2307	2%	0.4 D_{min}	90 s	201 s
C6288	2416	16.5%	0.4 D_{min}	1677 s	4138 s
C7552	3512	3.3%	0.4 D_{min}	320 s	683 s

content is minimal and the details of implementation are vastly different.

In essence, the proposed technique and the techniques in [2,5,8] are iterative relaxation approaches that involve a two-step optimization strategy. The first-step involves a delay budgeting step where optimal delays are computed for transistors/gates. The second step involves sizing transistors optimally under this "constant delay" model to achieve these delay budgets. The two steps are iteratively alternated until the solution converges, i. e., until the delay budgets calculated in the first step are exactly satisfied by the transistor sizes determined by the second step.

The primary features of the proposed approach are:

- It is computationally fast and is comparable to TILOS in its run-time behavior.
- It can be used for true transistor sizing as well as the relaxed problem of gate sizing. Additionally, the approach can easily incorporate wire-sizing [15].
- It can be adapted for more general delay models than the Elmore delay model [15].

The starting point for the proposed approach is a fast guess solution. This could be obtained, for example, from a circuit that has been optimized using TILOS to meet the given delay requirements. The proposed approach, as outlined earlier, is an iterative relaxation procedure that involves an alternating two-phase relaxed optimization sequence that is repeated iteratively until convergence is achieved. The two-phases in the proposed approach are:

- The **D-phase** where transistor sizes are assumed fixed and transistor delays are regarded as variable parameters. Irrespective of the delay model employed, this phase can be formulated as the dual of a min-cost network flow problem. Using $|V|$ to denote the number of transistors and $|E|$ the number of wires in the circuit, this step in our application has worst-case complexity of $O(|V|\,|E|\log(\log|V|))$ [7].
- The **W-phase** where transistor/gate delays are assumed fixed and their sizes are regarded as variable parameters. As long as the gate delay can be expressed as a separable function of the transistor sizes, this step can be solved as a Simple Monotonic Program (SMP) [11]. The complexity of SMP is similar to an all-pairs shortest path algorithm in a directed graph, [4,11], i. e., $O(|V|\,|E|)$.

The objective function for the problem is the minimization of circuit area. In the W-phase, this objective is addressed directly, and in the D-phase the objective is chosen to facilitate a move in the solution space in a direction that is known to lead to a reduction in the circuit area.

Applications

The primary application of the solution provided here is circuit and system optimization in automated VLSI design. The solution provided here can enable Electronic Design Automation (EDA) tools that take a holistic approach towards transistor sizing. This will in turn enable making custom circuit design flows more realizable in practice. The mechanics of some of the elements of the solution provided here especially the **D-phase** have been used to address other circuit optimization problems [14].

Open Problems

The related problem of Discrete gate sizing optimization matching gate sized to available gate sizes from a standard cell library is a provably hard optimization problem which could be aided by the development of efficient heuristics and probabilistic algorithms.

Experimental Results

A relative comparison of MINFLOTRANSIT with TILOS is provided in Table 1 for gate sizing of ISACS85 and mcnc91 benchmark circuits. As can be seen a significant performance improvement is observed with a tolerable loss in execution time.

Cross References

► Circuit Retiming
► Wire Sizing

Recommended Reading

1. Chen, C.P., Chu, C.N, Wong, D.F.: Fast and Exact Simultaneous Gate and Wire Sizing by Lagrangian Relax-ation. In: Proceedings of the 1998 IEEE/ACM International Conference on Computer-Aided Design, pp. 617–624. November 1998
2. Chen, H.Y., Kang, S.M.: icoach: A circuit optimiza-tion aid for cmos high-performance circuits. Intergr. VLSI. J. **10**(2), 185–212 (1991)
3. Conn, A.R., Coulman, P.K., Haring, R.A., Morrill, G.L., Visweshwariah, C., Wu, C.W.: JiffyTune: Circuit Optimization Using Time-Domain Sensitivities. IEEE Trans. Comput. Aided. Des. Integr. Circuits. Syst.**17**(12), 1292–1309 (1998)
4. Cormen, T.H., Leiserson, C.E., Rivest, R.L.: Introduction to algorithms. McGraw-Hill, New York (1990)
5. Dai, Z., Asada, K.: MOSIZ: A Two-Step Transistor Sizing Algorithm based on Optimal Timing Assignment Method for Multi-Stage Complex Gates. In: Proceedings of the 1989 Custom Integrated Circuits Conference, pp. 17.3.1–17.3.4. May 1989
6. Fishburn, J.P., Dunlop, A. E.: TILOS: A Posynomial Programming Approach to Transistor Sizing. In: Proceedings of the 1985 International Conference on Computer-Aided Design, pp. 326–328. Santa Clara, CA, November 1985
7. Goldberg, A.V., Grigoriadis, M.D., Tarjan, R.E.: Use of Dynamic Trees in a Network Simplex Algorithm for the Maximum Flow Problem. Math. Program. **50**(3), 277–290 (1991)
8. Grodstein, J., Lehman, E., Harkness, H., Grundmann, B., Watanabe, Y.: A delay model for logic synthesis of continuously sized networks. In: Proceedings of the 1995 International Conference on Computer-Aided Design, pp. 458–462. November 1995
9. Marple, D.P.: Performance Optimization of Digital VLSI Circuits. Technical Report CSL-TR-86-308, Stanford University, October 1986
10. Marple, D.P.: Transistor Size Optimization in the Tailor Layout System. In: Proceedings of the 26th ACM/IEEE Design Automation Conference, pp. 43–48. June 1989
11. Papaefthymiou, M.C.: Asymptotically Efficient Retiming under Setup and Hold Constraints. In: Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, pp. 288–295, November 1998
12. Sapatnekar, S.S., Rao, V.B., Vaidya, P.M., Kang, S.M.: An Exact Solution to the Transistor Sizing Problem for CMOS Circuits Using Convex Optimization. IEEE Trans. Comput. Aided. Des. **12**(11), 1621–1634 (1993)
13. Shyu, J.M., Sangiovanni-Vincentelli, A.L., Fishburn, J.P., Dunlop, A.E.: Optimization-based Transistor Sizing. IEEE J. Solid. State. Circuits. **23**(2), 400–409 (1988)
14. Sundararajan, V., Parhi, K.: Low Power Synthesis of Dual Threshold Voltage CMOS VLSI Circuits. In: Proceedings of the International Symposium on Low Power Electronics and Design. pp. 139-144 (1999)
15. Sundararajan, V., Sapatnekar, S.S., Parhi, K.K.: Fast and exact transistor sizing based on iterative relaxation. Computer-Aided Design of Integrated Circuits and Systems, IEEE Trans. **21**(5),568–581 (2002)

General Equilibrium

2002; Deng, Papadimitriou, Safra

LI-SHA HUANG
Department of Computer Science and Technology, Tsinghua University, Beijing, Beijing, China

Keywords and Synonyms

Competitive market equilibrium

Problem Definition

This problem is concerned with the computational complexity of finding an exchange market equilibrium. The exchange market model consists of a set of agents, each with an initial endowment of commodities, interacting through a market, trying to maximize each's utility function. The equilibrium prices are determined by a clearance condition. That is, all commodities are bought, collectively, by all the utility maximizing agents, subject to their budget constraints (determined by the values of their initial endowments of commodities at the market price). The work of Deng, Papadimitriou and Safra [3] studies the complexity, approximability, inapproximability, and communication complexity of finding equilibrium prices. The work shows the NP-hardness of approximating the equilibrium in a market with indivisible goods. For markets with divisible goods and linear utility functions, it develops a pseudo-polynomial time algorithm for computing an ϵ-equilibrium. It also gives a communication complexity lower bound for computing Pareto allocations in markets with non-strictly concave utility functions.

Market Model

In a pure exchange economy, there are m traders, labeled by $i = 1, 2, \ldots, m$, and n types of commodities, labeled by $j = 1, 2, \ldots, n$. The commodities could be divisible or indivisible. Each trader i comes to the market with initial endowment of commodities, denoted by a vector $w_i \in \mathbb{R}^n_+$, whose j-th entry is the amount of commodity j held by trader i.

Associate each trader i a *consumption set* X_i to represents the set of possible commodity bundles for him. For example, when there are n_1 divisible commodities and $(n - n_1)$ indivisible commodities, X_i can be $\mathbb{R}^{n_1}_+ \times \mathbb{Z}^{n-n_1}_+$. Each trader has a utility function $X_i \mapsto \mathbb{R}_+$ to present his utility for a bundle of commodities. Usually, the utility function is required to be concave and nondecreasing.

In the market, each trader acts as both a buyer and a seller to maximize his utility. At a certain price $p \in \mathbb{R}^n_+$, trader i is is solving the following optimization problem, under his budget constraint:

$$\max \quad u_i(x_i) \quad s.t. \quad x_i \in X_i \text{ and } \langle p, x_i \rangle \le \langle p, w_i \rangle .$$

Definition 1 An equilibrium in a pure exchange economy is a price vector $\bar{p} \in \mathbb{R}^n_+$ and bundles of commodities $\{\bar{x}_i \in \mathbb{R}^n_+, i = 1, \ldots, m\}$, such that

$$\bar{x}_i \in \operatorname{argmax}\{u_i(x_i) | x_i \in X_i \text{ and } \langle x_i, \bar{p} \rangle \le \langle w_i, \bar{p} \rangle\}, \quad \forall 1 \le i \le m$$

$$\sum_{i=1}^{m} \bar{x}_{ij} \le \sum_{i=1}^{m} w_{ij}, \forall 1 \le j \le n .$$

The concept of approximate equilibrium was introduced in [3]:

Definition 2 ([3]) An ϵ-approximate equilibrium in an exchange market is a price vector $\bar{p} \in \mathbb{R}^n_+$ and bundles of goods $\{\bar{x}_i \in \mathbb{R}^n_+, i = 1, \ldots, m\}$, such that

$$u_i(\bar{x}_i) \ge \frac{1}{1+\epsilon} \max\{u_i(x_i) | x_i \in X_i, \langle x_i, \bar{p} \rangle \le \langle w_i, \bar{p} \rangle\}, \forall i \quad (1)$$

$$\langle \bar{x}_i, \bar{p} \rangle \le (1+\epsilon)\langle w_i, \bar{p} \rangle, \forall i \quad (2)$$

$$\sum_{i=1}^{m} \bar{x}_{ij} \le (1+\epsilon) \sum_{i=1}^{m} w_{ij}, \forall j . \quad (3)$$

Key Results

A linear market is a market in which all the agents have linear utility functions. The deficiency of a market is the smallest $\epsilon \ge 0$ for which an ϵ-approximate equilibrium exists.

Theorem 1 *The deficiency of a linear market with indivisible goods is NP-hard to compute, even if the number of agents is two. The deficiency is also NP-hard to approximate within 1/3.*

Theorem 2 *There is a polynomial-time algorithm for finding an equilibrium in linear markets with bounded number of divisible goods. Ditto for a polynomial number of agents.*

Theorem 3 *If the number of goods is bounded, there is a polynomial-time algorithm which, for any linear indivisible market for which a price equilibrium exists, and for any $\epsilon > 0$, finds an ϵ-approximate equilibrium.*

If the utility functions are strictly concave and the equilibrium prices are broadcasted to all agents, the equilibrium allocation can be computed distributely without any communication, since each agent's basket of goods is uniquely determined. However, if the utility functions are not strictly concave, e. g. linear functions, communications are needed to coordinate the agents' behaviors.

Theorem 4 *Any protocol with binary domains for computing Pareto allocations of m agents and n divisible commodities with concave utility functions (resp. ϵ-Pareto allocations for indivisible commodities, for any $\epsilon < 1$) must have market communication complexity $\Omega(m \log(m+n))$ bits.*

Applications

This concept of market equilibrium is the outcome of a sequence of efforts trying to fully understand the laws that govern human commercial activities, starting with the "invisible hand" of Adam Smith, and finally, the mathematical conclusion of Arrow and Debreu [1] that there exists a set of prices that bring supply and demand into equilibrium, under quite general conditions on the agent utility functions and their optimization behavior.

The work of Deng, Papadimitriou and Safra [3] explicitly called for an algorithmic complexity study of the problem, and developed interesting complexity results and approximation algorithms for several classes of utility functions. There has since been a surge of algorithmic study for the computation of the price equilibrium problem with continuous variables, discovering and rediscovering polynomial time algorithms for many classes of utility functions, see [2,4,5,6,7,8,9].

Significant progress has been made in the above directions but only as a first step. New ideas and methods have already been invented and applied in reality. The next significant step will soon manifest itself with many active studies in microeconomic behavior analysis for E-commercial markets. Nevertheless the algorithmic analytic foundation in [3] will be an indispensable tool for further development in this reincarnated exciting field.

Open Problems

The most important open problem is what is the computational complexity for finding the equilibrium price, as guaranteed by the Arrow–Debreu theorem. To the best of the author's knowledge, only the markets whose set of equilibria is convex can be solved in polynomial time with current techniques. And approximating equilibria in some markets with disconnected set of equilibria, e. g. Leontief economies, are shown to be PPAD-hard. Is the convexity or (weakly) gross substitutability a necessary condition for a market to be polynomial-time solvable?

Second, how to handle the dynamic case is especially interesting in theory, mathematical modeling, and algorithmic complexity as bounded rationality. Great progress must be made in those directions for any theoretical work to be meaningful in practice.

Third, incentive compatible mechanism design protocols for the auction models have been most actively studied recently, especially with the rise of E-Commerce. Especially at this level, a proper approximate version of the equilibrium concept handling price dynamics should be especially important.

Cross References

► Complexity of Core
► Leontief Economy Equilibrium
► Non-approximability of Bimatrix Nash Equilibria

Recommended Reading

1. Arrow, K.J., Debreu, G.: Existence of an equilibrium for a competitive economy. Econometrica **22**(3), 265–290 (1954)
2. Codenotti, B., McCune, B., Varadarajan, K.: Market equilibrium via the excess demand function. In: Proceedings STOC'05, pp. 74–83. ACM, Baltimore (2005)
3. Deng, X., Papadimitriou, C., Safra, S.: On the complexity of price equilibria. J. Comput. Syst. Sci. **67**(2), 311–324 (2002)
4. Devanur, N.R., Papadimitriou, C.H., Saberi, A., Vazirani, V.V.: Market equilibria via a primal-dual-type algorithm. In: Proceedings of FOCS'02, pp. 389–395. IEEE Computer Society, Vancouver (2002)
5. Eaves, B.C.: Finite solution for pure trade markets with Cobb-Douglas utilities, Math. Program. Study **23**, 226–239 (1985)
6. Garg, R., Kapoor, S.: Auction algorithms for market equilibrium, In: Proceedings of STOC'04, pp. 511–518. ACM, Chicago (2004)
7. Jain, K.: A polynomial time algorithm for computing the Arrow-Debreu market equilibrium for linear utilities. In: Proceeding of FOCS'04, pp. 286–294. IEEE Computer Society, Rome (2004)
8. Nenakhov, E., Primak, M.: About one algorithm for finding the solution of the Arrow-Debreu Model. Kibernetica **3**, 127–128 (1983)
9. Ye, Y.: A path to the Arrow-Debreu competitive market equilibrium, Math. Program. **111**(1–2), 315–348 (2008)

Generalized Steiner Network

2001; Jain

JULIA CHUZHOY
Toyota Technological Institute, Chicago, IL, USA

Keywords and Synonyms

Survivable network design

Problem Definition

The generalized Steiner network problem is a network design problem, where the input consists of a graph together with a collection of connectivity requirements, and the goal is to find the cheapest subgraph meeting these requirements.

Formally, the input to the generalized Steiner network problem is an undirected multigraph $G = (V, E)$, where each edge $e \in E$ has a non-negative cost $c(e)$, and for each pair of vertices $i, j \in V$, there is a connectivity requirement $r_{i,j} \in \mathbb{Z}$. A feasible solution is a subset $E' \subseteq E$ of edges, such that every pair $i, j \in V$ of vertices is connected by at least $r_{i,j}$ edge-disjoint path in graph $G' = (V, E')$.

The generalized Steiner network problem asks to find a solution E' of minimum cost $\sum_{e \in E'} c(e)$.

This problem generalizes several classical network design problems. Some examples include minimum spanning tree, Steiner tree and Steiner forest. The most general special case for which a 2-approximation was previously known is the Steiner forest problem [1,4].

Williamson et al. [8] were the first to show a non-trivial approximation algorithm for the generalized Steiner network problem, achieving a $2k$-approximation, where $k = \max_{i,j \in V}\{r_{i,j}\}$. This result was improved to $O(\log k)$-approximation by Goemans et al. [3].

Key Results

The main result of [6] is a factor-2 approximation algorithm for the generalized Steiner network problem. The techniques used in the design and the analysis of the algorithm seem to be of independent interest.

The 2-approximation is achieved for a more general problem, defined as follows. The input is a multigraph $G = (V, E)$ with costs $c(\cdot)$ on edges, and connectivity requirement function $f : 2^V \to \mathbb{Z}$. Function f is weakly submodular, i. e., it has the following properties:

1. $f(V) = 0$.
2. For all $A, B \subseteq V$, at least one of the following two conditions holds:
   - $f(A) + f(B) \le f(A \setminus B) + f(B \setminus A)$.
   - $f(A) + f(B) \le f(A \cap B) + f(A \cup B)$.

For any subset $S \subseteq V$ of vertices, let $\delta(S)$ denote the set of edges with exactly one endpoint in S. The goal is to find a minimum-cost subset of edges $E' \subseteq E$, such that for every subset $S \subseteq V$ of vertices, $|\delta(S) \cap E'| \ge f(S)$.

This problem can be equivalently expressed as an integer program. For each edge $e \in E$, let x_e be the indicator variable of whether e belongs to the solution.

$$\text{(IP)} \qquad \min \sum_{e \in E} c(e) x_e$$

subject to:

$$\sum_{e \in \delta(S)} x_e \ge f(S) \qquad \forall S \subseteq V \qquad (1)$$

$$x_e \in \{0, 1\} \qquad \forall e \in E \qquad (2)$$

It is easy to see that the generalized Steiner network problem is a special case of (IP), where for each $S \subseteq V$, $f(S) = \max_{i \in S, j \notin S}\{r_{i,j}\}$.

Techniques

The approximation algorithm uses the LP-rounding technique. The initial linear program (LP) is obtained from (IP) by replacing the integrality constraint (2) with:

$$0 \le x_e \le 1 \qquad \forall e \in E \qquad (3)$$

It is assumed that there is a separation oracle for (LP). It is easy to see that such an oracle exists if (LP) is obtained from the generalized Steiner network problem. The key result used in the design and the analysis of the algorithm is summarized in the following theorem.

Theorem 1 *In any basic solution of (LP), there is at least one edge $e \in E$ with $x_e \ge 1/2$.*

The approximation algorithm works by iterative LP-rounding. Given a basic optimal solution of (LP), let $E^* \subseteq E$ be the subset of edges e with $x_e \ge 1/2$. The edges of E^* are removed from the graph (and are eventually added to the solution), and the problem is then solved recursively on the residual graph, by solving (LP) on $G^* = (V, E \setminus E^*)$, where for each subset $S \subseteq V$, the new requirement is $f(S) - |\delta(S) \cap E^*|$. The main observation that leads to factor-2 approximation is the following: if E' is a 2-approximation for the residual problem, then $E' \cup E^*$ is a 2-approximation for the original problem.

Given any solution to (LP), set $S \subseteq V$ is called *tight* iff constraint (1) holds with equality for S. The proof of Theorem 1 involves constructing a large *laminar family* of tight sets (a family where for every pair of sets, either one set contains the other, or the two sets are disjoint). After that a clever accounting scheme that charges edges to the sets of the laminar family is used to show that there is at least one edge $e \in E$ with $x_e \ge 1/2$.

Applications

Generalized Steiner network is a very basic and natural network design problem that has many applications in different areas, including the design of communication networks, VLSI design and vehicle routing. One example is the design of survivable communication networks, which remain functional even after the failure of some network components (see [5] for more details).

Open Problems

The 2-approximation algorithm of Jain [6] for generalized Steiner network is based on LP-rounding, and it has high running time. It would be interesting to design a combinatorial approximation algorithm for this problem.

It is not known whether a better approximation is possible for generalized Steiner network. Very few hardness of approximation results are known for this type of problems. The best current hardness factor stands on 1.01063 [2], and this result is valid even for the special case of Steiner tree.

Cross References

► Steiner Forest
► Steiner Trees

Recommended Reading

1. Agrawal A., Klein P., Ravi R.: When Trees Collide: An Approximation Algorithm for the Generalized Steiner Problem on Networks. J. SIAM Comput. **24**(3), 440–456 (1995)
2. Chlebik M., Chlebikova J.: Approximation Hardness of the Steiner Tree Problem on Graphs. In: 8th Scandinavian Workshop on Algorithm Theory. Number 2368 in LNCS, pp. 170–179, (2002)
3. Goemans M.X., Goldberg A.V., Plotkin S.A., Shmoys D.B., Tardos É., Williamson D.P.: Improved Approximation Algorithms for Network Design Problems. In: Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 223–232. (1994)
4. Goemans M.X., Williamson D.P.: A General Approximation Technique for Constrained Forest Problems. SIAM J. Comput. **24**(2), 296–317 (1995)
5. Grötschel M., Monma C.L., Stoer M.: Design of Survivable Networks. In: Network Models, Handbooks in Operations Research and Management Science. North Holland Press, Amsterdam, (1995)
6. Jain K.: A Factor 2 Approximation Algorithm for the Generalized Steiner Network Problem. Combinatorica **21**(1), 39–60 (2001)
7. Vazirani V.V.: Approximation Algorithms. Springer, Berlin (2001)
8. Williamson D.P., Goemans M.X., Mihail M., Vazirani V.V.: A Primal-Dual Approximation Algorithm for Generalized Steiner Network Problems. Combinatorica **15**(3), 435–454 (1995)

Generalized Two-Server Problem

2006; Sitters, Stougie

René A. Sitters
Department of Mathematics and Computer Science, Eindhoven University of Technology, Eindhoven, The Netherlands

Keywords and Synonyms

CNN-problem

Problem Definition

In the *generalized two-server problem* we are given two servers: one moving in a metric space $\mathbb{X}$ and one moving in a metric space $\mathbb{Y}$. They are to serve requests $r \in \mathbb{X} \times \mathbb{Y}$ which arrive one by one. A request $r = (x, y)$ is served by moving either the $\mathbb{X}$-server to point x or the $\mathbb{Y}$-server to point y. The decision as to which server to move to the next request is irrevocable and has to be taken without any knowledge about future requests. The objective is to minimize the total distance traveled by the two servers.

On-line Routing Problems

The generalized two-server problem belongs to a class of routing problems called *metrical service systems* [5,10]. Such a system is defined by a metric space $\mathbb{M}$ of all possible system configurations, an initial configuration C_0, and a set $\mathcal{R}$ of possible requests, where each request $r \in \mathcal{R}$ is a subset of $\mathbb{M}$. Given a sequence, $r_1, r_2 \ldots, r_n$, of requests, a feasible solution is a sequence, $C_1, C_2, \ldots, C_n$, of configurations such that $C_i \in r_i$ for all $i \in \{1, \ldots, n\}$.

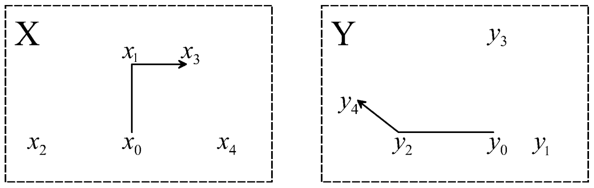
When we model the generalized two-server problem as a metrical service system we have $\mathbb{M} = \mathbb{X} \times \mathbb{Y}$ and $\mathcal{R} = \{\{x \times \mathbb{Y}\} \cup \{\mathbb{X} \times y\} | x \in \mathbb{X}, y \in \mathbb{Y}\}$. In the classical *two-server problem* both servers move in the same space and receive the same requests, i. e., $\mathbb{M} = \mathbb{X} \times \mathbb{X}$ and $\mathcal{R} = \{\{x \times \mathbb{X}\} \cup \{\mathbb{X} \times x\} | x \in \mathbb{X}\}$.

The performance of algorithms for on-line optimization problems is often measured using *competitive analysis*. We say that an algorithm is α*-competitive* ($\alpha \geq 1$) for some minimization problem if for every possible instance the cost of the algorithm's solution is at most α times the cost of an optimal solution for the instance.

A standard algorithm that performs provably well for several elementary routing problems is the so-called *work function algorithm* [2,6,8]; after each request the algorithm moves to a configuration with low cost and which is not too far from the current configuration. More precisely: If the system's configuration after serving a sequence σ is C and $r \subseteq \mathbb{M}$ is the next request, then the work function algorithm with parameter $\lambda \geq 1$ moves to a configuration $C' \in r$ that minimizes

$$\lambda W_{\sigma,r}(C') + d(C, C') ,$$

where $d(C, C')$ is the distance between configurations C and C', and $W_{\sigma,r}(C')$ is the cost of an optimal solution that



Generalized Two-Server Problem, Figure 1
In this example both servers move in the plane and start from the configuration (x_0, y_0). The $\mathbb{X}$-server moves through requests 1 and 3, and the $\mathbb{Y}$-server takes care of requests 2 and 4. The cost of this solution is the sum of the path-lengths

serves all requests (in order) in σ plus request r with the restriction that it ends in configuration C'.

Key Results

The main result in [11] is a sufficient condition for a metrical service system to have a constant-competitive algorithm. Additionally, the authors show that this condition holds for the generalized two-server problem.

For a fixed metrical service system S with metric space $\mathbb{M}$, denote by $A(C, \sigma)$ the cost of algorithm A on input sequence σ, starting in configuration C. Let $\textsc{Opt}(C, \sigma)$ be the cost of the corresponding optimal solution. We say that a path T in $\mathbb{M}$ *serves* a sequence σ if it visits all requests in order. Hence, a feasible path is a path that serves the sequence and starts in the initial configuration.

Paths T_1 and T_2 are said to be *independent* if they are far apart in the following way: $|T_1| + |T_2| < d(C_1^s, C_2^t) + d(C_2^s, C_1^t)$, where C_i^s and C_i^t are, respectively, the start and end point of path $T_i (i \in \{1, 2\})$. Notice, for example, that two intersecting paths are not independent.

Theorem 1 *Let S be a metrical service system with metric space $\mathbb{M}$. Suppose there exists an algorithm A and constants $\alpha \geq 1, \beta \geq 0$ and $m \geq 2$ such that for any point $C \in \mathbb{M}$, sequence σ and pairwise independent paths $T_1, T_2, \ldots, T_m$ that serve σ*

$$A(C, \sigma) \leq \alpha \textsc{Opt}(C, \sigma) + \beta \sum_{i=1}^{m} |T_i| \,. \tag{1}$$

Then there exists an algorithm B that is constant competitive for S.

The proof in [11] of the theorem above provides an explicit formulation of B. This algorithm combines algorithm A with the work function algorithm and operates in phases. In each phase, it applies algorithm A until its cost becomes too large compared to the optimal cost. Then, it makes one step of the work function algorithm and a new phase starts. In each phase algorithm A makes a restart, i. e., it takes the final configuration of the previous phase as the initial configuration, whereas the work function algorithm remembers the whole request sequence.

For the generalized two-server problem the so-called *balance algorithm* satisfies condition (1). This algorithm stores the cumulative costs of the two servers and with each request it moves the server that minimizes the maximum of the two new values. The balance algorithm itself is not constant competitive but Theorem 1 says that, if we combine it in a clever way with the work function algorithm, then we get an algorithm that is constant competitive.

Applications

A set of metrical service systems can be combined to get what is called in [9] the *sum system*. A request of the sum system consists of one request for each system and to serve it we need to serve at least one of the individual requests. The generalized two-server problem should be considered as one of the simplest sum systems since the two individual problems are completely trivial: there is one server and each request consists of a single point.

Sum systems are particularly interesting to model systems for information storage and retrieval. To increase stability or efficiency one may store copies of the same information in multiple systems (e. g. databases, hard disks). To retrieve one piece of information we may read it from any system. However, to read information it may be necessary to change the configuration of the system. For example, if the database is stored in a binary search tree, then it is efficient to make on-line changes to the structure of the tree, i. e., to use dynamic search trees [12].

Open Problems

A proof that the work function algorithm is competitive for the generalized two-server problem (as conjectured in [9] and [11]) is still lacking. Also, a randomized algorithm with a smaller competitive ratio than that of [11] is not known. No results (except for a lower bound) are known for the generalized problem with more than two servers. It is not even clear if the work function algorithm may be competitive here.

There are systems for which the work function algorithm is not competitive. It would be interesting to have a non-trivial property that implies competitiveness of the work function algorithm.

Cross References

▶ Algorithm DC-Tree for k Servers on Trees
▶ Metrical Task Systems
▶ Online Paging and Caching
▶ Work-Function Algorithm for k Servers

Recommended Reading

1. Borodin, A., El-Yaniv, R.: Online computation and competitive analysis. Cambridge University Press, Cambridge (1998)
2. Burley, W.R.: Traversing layered graphs using the work function algorithm. J. Algorithms **20**, 479–511 (1996)
3. Chrobak, M.: Sigact news online algorithms column 1. ACM SIGACT News **34**, 68–77 (2003)
4. Chrobak, M., Karloff, H., Payne, T.H., Vishwanathan, S.: New results on server problems. SIAM J. Discret. Math. **4**, 172–181 (1991)

5. Chrobak, M., Larmore, L.L.: Metrical service systems: Deterministic strategies. Tech. Rep. UCR-CS-93-1, Department of Computer Science, Univ. of California at Riverside (1992)
6. Chrobak, M., Sgall, J.: The weighted 2-server problem. Theor. Comput. Sci. **324**, 289–312 (2004)
7. Fiat, A., Ricklin, M.: Competitive algorithms for the weighted server problem. Theor. Comput. Sci. **130**, 85–99 (1994)
8. Koutsoupias, E., Papadimitriou, C.H.: On the *k*-server conjecture. J. ACM **42**, 971–983 (1995)
9. Koutsoupias, E., Taylor, D.S.: The CNN problem and other k-server variants. Theor. Comput. Sci. **324**, 347–359 (2004)
10. Manasse, M.S., McGeoch, L.A., Sleator, D.D.: Competitive algorithms for server problems. J. Algorithms **11**, 208–230 (1990)
11. Sitters, R.A., Stougie, L.: The generalized two-server problem. J. ACM **53**, 437–458 (2006)
12. Sleator, D.D., Tarjan, R.E.: Self-adjusting binary search trees. J. ACM **32**, 652–686 (1985)

Generalized Vickrey Auction

1995; Varian

Makoto Yokoo
Department of Information Science and Electrical Engineering, Kyushu University, Nishi-ku, Japan

Keywords and Synonyms

Generalized Vickrey auction; GVA;
Vickrey–Clarke–Groves mechanism; VCG

Problem Definition

Auctions are used for allocating goods, tasks, resources, etc. Participants in an auction include an auctioneer (usually a seller) and bidders (usually buyers). An auction has well-defined rules that enforce an agreement between the auctioneer and the winning bidder. Auctions are often used when a seller has difficulty in estimating the value of an auctioned good for buyers.

The Generalized Vickrey Auction protocol (GVA) [5] is an auction protocol that can be used for combinatorial auctions [3] in which multiple items/goods are sold simultaneously. Although conventional auctions sell a single item at a time, combinatorial auctions sell multiple items/goods. These goods may have interdependent values, e. g., these goods are complementary/substitutable and bidders can bid on any combination of goods. In a combinatorial auction, a bidder can express complementary/substitutable preferences over multiple bids. By taking into account complementary/substitutable preferences, the participants' utilities and the revenue of the seller can be increased. The GVA is one instance of the Clarke mechanism [2,4]. It is also called the Vickrey–Clarke–Groves mechanism (VCG). As its name suggests, it is a generalized version of the well-known Vickrey (or second-price) auction protocol [6], proposed by an American economist W. Vickrey, a 1996 Nobel Prize winner.

Assume there is a set of bidders $N = \{1, 2, \dots, n\}$ and a set of goods $M = \{1, 2, \dots, m\}$. Each bidder i has his/her preferences over a bundle, i. e., a subset of goods $B \subseteq M$. Formally, this can be modeled by supposing that bidder i privately observes a parameter, or signal, θ_i, which determines his/her preferences. The parameter θ_i is called the *type* of bidder i. A bidder is assumed to have a *quasilinear*, *private value* defined as follows.

Definition 1 (Utility of a Bidder) The utility of bidder i, when i obtains $B \subseteq M$ and pays p_i, is represented as $v(B, \theta_i) - p_i$.

Here, the valuation of a bidder is determined independently of other bidders' valuations. Also, the utility of a bidder is linear in terms of the payment. Thus, this model is called a quasilinear, private value model.

Definition 2 (Incentive Compatibility) An auction protocol is (dominant-strategy) *incentive compatible* (or *strategy-proof*) if declaring the true type/evaluation values is a dominant strategy for each bidder, i. e., an optimal strategy regardless of the actions of other bidders.

A combination of dominant strategies of all bidders is called a *dominant-strategy equilibrium*.

Definition 3 (Individual Rationality) An auction protocol is *individually rational* if no participant suffers any loss in a dominant-strategy equilibrium, i. e., the payment never exceeds the evaluation value of the obtained goods.

Definition 4 (Pareto Efficiency) An auction protocol is *Pareto efficient* when the sum of all participants' utilities (including that of the auctioneer), i. e., the social surplus, is maximized in a dominant-strategy equilibrium.

The goal is to design an auction protocol that is incentive compatible, individually rational, and Pareto efficient. It is clear that individual rationality and Pareto efficiency are desirable. Regarding the incentive compatibility, the *revelation principle* states that in the design of an auction protocol, it is possible to restrict attention only to incentive compatible protocols without loss of generality [4]. In other words, if a certain property (e. g., Pareto efficiency) can be achieved using some auction protocol in a dominant-strategy equilibrium, then the property can also be achieved using an incentive-compatible auction protocol.

Key Results

A *feasible* allocation is defined as a vector of n bundles $\vec{B} = \langle B_1, \dots, B_n \rangle$, where $\bigcup_{j \in N} B_j \subseteq M$ and for all $j \neq j', B_j \cap B_{j'} = \emptyset$ hold.

The GVA protocol can be described as follows.

1. Each bidder i declares his/her type $\hat{\theta}_i$, which can be different from his/her true type θ_i.
2. The auctioneer chooses an optimal allocation $\vec{B}^*$ according to the declared types. More precisely, the auctioneer chooses $\vec{B}^*$ defined as follows:

$$\vec{B}^* = \arg\max_{\vec{B}} \sum_{j \in N} v\left(B_j, \hat{\theta}_j\right).$$

3. Each bidder i pays p_i, which is defined as follows ($B_j^{\sim i}$ and B_j^* are the jth element of $\vec{B}^{\sim i}$ and $\vec{B}^*$, respectively):

$$p_i = \sum_{j \in N \setminus \{i\}} v\left(B_j^{\sim i}, \hat{\theta}_j\right) - \sum_{j \in N \setminus \{i\}} v\left(B_j^*, \hat{\theta}_j\right),$$
$$\text{where } \vec{B}^{\sim i} = \arg\max_{\vec{B}} \sum_{j \in N \setminus \{i\}} v\left(B_j, \hat{\theta}_j\right). \qquad (1)$$

The first term in Eq. (1) is the social surplus when bidder i does not participate. The second term is the social surplus except bidder i when i does participate. In the GVA, the payment of bidder i can be considered as the decreased amount of the other bidders' social surplus resulting from his/her participation.

A description of how this protocol works is given below.

Example 1 Assume there are two goods a and b, and three bidders, 1, 2, and 3, whose types are θ_1, θ_2, and θ_3, respectively. The evaluation value for a bundle $v(B, \theta_i)$ is determined as follows.

	$\{a\}$	$\{b\}$	$\{a, b\}$
θ_1	\$6	\$0	\$6
θ_2	\$0	\$0	\$8
θ_3	\$0	\$5	\$5

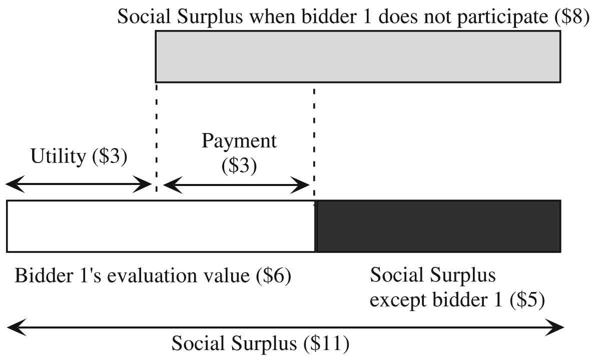
Here, bidder 1 wants good a only, and bidder 3 wants good b only. Bidder 2's utility is all-or-nothing, i. e., he/she wants both goods at the same time and having only one good is useless.

Assume each bidder i declares his/her true type θ_i. The optimal allocation is to allocate good a to bidder 1 and b to bidder 3, i. e., $\vec{B}^* = \langle \{a\}, \{\}, \{b\} \rangle$. The payment of bidder 1 is calculated as follows. If bidder 1 does not participate, the optimal allocation would have been allocating both items to bidder 2, i. e., $\vec{B}^{\sim 1} = \langle \{\}, \{a, b\}, \{\} \rangle$ and the social surplus, i. e., $\sum_{j \in N \setminus \{1\}} v\left(B_j^{\sim 1}, \hat{\theta}_j\right)$ is equal to \$8. When bidder 1 does participate, bidder 3 obtains $\{b\}$, and the social surplus except for bidder 1, i. e., $\sum_{j \in N \setminus \{1\}} v\left(B_j^*, \hat{\theta}_j\right)$, is 5. Therefore, bidder 1 pays the difference \$8 − \$5 = \$3. The obtained utility of bidder 1 is \$6 − \$3 = \$3. The payment of bidder 3 is calculated as \$8 − \$6 = \$2.

The intuitive explanation of why truth telling is the dominant strategy in the GVA is as follows. In the GVA, goods are allocated so that the social surplus is maximized. In general, the utility of society as a whole does not necessarily mean maximizing the utility of each participant. Therefore, each participant might have an incentive for lying if the group decision is made so that the social surplus is maximized.

However, the payment of each bidder in the GVA is cleverly determined so that the utility of each bidder is maximized when the social surplus is maximized. Figure 1 illustrates the relationship between the payment and utility of bidder 1 in Example 1. The payment of bidder 1 is defined as the difference between the social surplus when bidder 1 does not participate (i. e., the length of the upper shaded bar) and the social surplus except bidder 1 when bidder 1 does participate (the length of the lower black bar), i. e., \$8 − \$5 = \$3.

On the other hand, the utility of bidder 1 is the difference between the evaluation value of the obtained item and the payment, which equals \$6 − \$3 = \$3. This amount is equal to the difference between the total length of the lower bar and the upper bar. Since the length of the upper bar is determined independently of bidder 1's declaration, bidder 1 can maximize his/her utility by maximizing the length of the lower bar. However, the length of the lower bar represents the social surplus. Thus, bidder 1 can maximize his/her utility when the social surplus is maximized.


Social Surplus when bidder 1 does not participate ($8)
Utility ($3)
Payment ($3)
Bidder 1's evaluation value ($6)
Social Surplus except bidder 1 ($5)
Social Surplus ($11)

Generalized Vickrey Auction, Figure 1
Utilities and Payments in the GVA

Therefore, bidder 1 does not have an incentive for lying since the group decision is made so that the social surplus is maximized.

Theorem 1 *The GVA is incentive compatible.*

Proof Since the utility of bidder i is assumed to be quasi-linear, it can be represented as

$$\begin{aligned} v(B_i, \theta_i) - p_i &= v(B_i, \theta_i) \\ &\quad - \left[\sum_{j \in N \setminus \{i\}} v\left(B_j^{\sim i}, \hat{\theta}_j\right) - \sum_{j \in N \setminus \{i\}} v\left(B_j^*, \hat{\theta}_j\right) \right] \\ &= \left[v(B_i, \theta_i) + \sum_{j \in N \setminus \{i\}} v\left(B_j^*, \hat{\theta}_j\right) \right] \\ &\quad - \sum_{j \in N \setminus \{i\}} v\left(B_j^{\sim i}, \hat{\theta}_j\right) \end{aligned} \qquad (2)$$

The second term in Eq. (2) is determined independently of bidder i's declaration. Thus, bidder 1 can maximize his/her utility by maximizing the first term. However, $\vec{B}^*$ is chosen so that $\sum_{j \in N} v\left(B_j, \hat{\theta}_j\right)$ is maximized. Therefore, bidder i can maximize his/her utility by declaring $\hat{\theta}_i = \theta_i$, i. e., by declaring his/her true type. □

Theorem 2 *The GVA is individually rational.*

Proof This is clear from Eq. (2), since the first term is always larger than (or at least equal to) the second term. □

Theorem 3 *The GVA is Pareto efficient.*

Proof From Theorem 1, truth telling is a dominant-strategy equilibrium. From the way of choosing the allocation, the social surplus is maximized if all bidders declare their true types. □

Applications

The GVA can be applied to combinatorial auctions, which have lately attracted considerable attention [3]. The US Federal Communications Commission has been conducting auctions for allocating spectrum rights. Clearly, there exist interdependencies among the values of spectrum rights. For example, a bidder may desire licenses for adjoining regions simultaneously, i. e., these licenses are complementary. Thus, the spectrum auctions is a promising application field of combinatorial auctions and have been a major driving force for activating the research on combinatorial auctions.

Open Problems

Although the GVA has these good characteristics (Pareto efficiency, incentive compatibility, and individual rationality), these characteristics cannot be guaranteed when bidders can submit *false-name* bids. Furthermore, [1] pointed out several other limitations such as vulnerability to the collusion of the auctioneer and/or losers.

Also, to execute the GVA, the auctioneer must solve a complicated optimization problem. Various studies have been conducted to introduce search techniques, which were developed in the artificial intelligence literature, for solving this optimization problem [3].

Cross References

► False-Name-Proof Auction

Recommended Reading

1. Ausubel, L.M., Milgrom, P.R.: Ascending auctions with package bidding. Front. Theor. Econ. **1**(1) Article 1 (2002)
2. Clarke, E.H., Multipart pricing of public goods. Publ. Choice **2**, 19–33 (1971)
3. Cramton, P., Steinberg, R., Shoham, Y. (eds.): Combinatorial Auctions. MIT Press, Cambridge (2005)
4. Mas-Colell, A., Whinston, M.D., Green, J.R.: Microeconomic Theory. Oxford University Press, Oxford (1995)
5. Varian, H.R.: Economic mechanism design for computerized agents. In: Proceedings of the 1st Usenix Workshop on Electronic Commerce, 1995
6. Vickrey, W.: Counter speculation, auctions, and competitive sealed tenders. J. Financ. **16**, 8–37 (1961)

Geographic Routing

2003; Kuhn, Wattenhofer, Zollinger

AARON ZOLLINGER
Department of Electrical Engineering and Computer Science, University of California at Berkeley, Berkeley, CA, USA

Keywords and Synonyms

Directional routing; Geometric routing; Location-based routing; Position-based routing

Problem Definition

Geographic routing is a type of routing particularly well suited for dynamic ad hoc networks. Sometimes also called directional, geometric, location-based, or position-based routing, it is based on two principal assumptions. First, it is assumed that every node knows its own and its network neighbors' positions. Second, the source of a message is assumed to be informed about the position of the destination. Geographic routing is defined on a Euclidean graph, that is a graph whose nodes are embedded in the Euclidean plane. Formally, geographic ad hoc routing algorithms can be defined as follows:

Definition 1 (Geographic Ad Hoc Routing Algorithm) Let $G = (V, E)$ be a Euclidean graph. The task of a geographic ad hoc routing algorithm $\mathcal{A}$ is to transmit a message from a source $s \in V$ to a destination $t \in V$ by sending packets over the edges of G while complying with the following conditions:

- All nodes $v \in V$ know their geographic positions as well as the geographic positions of all their neighbors in G.
- The source s is informed about the position of the destination t.
- The control information which can be stored in a packet is limited by $O(\log n)$ bits, that is, only information about a constant number of nodes is allowed.
- Except for the temporary storage of packets before forwarding, a node is not allowed to maintain any information.

Geographic routing is particularly interesting, as it operates without any routing tables whatsoever. Furthermore, once the position of the destination is known, all operations are strictly local, that is, every node is required to keep track only of its direct neighbors. These two factors—absence of necessity to keep routing tables up to date and independence of remotely occurring topology changes—are among the foremost reasons why geographic routing is exceptionally suitable for operation in ad hoc networks. Furthermore, in a sense, geographic routing can be considered a lean version of source routing appropriate for dynamic networks: While in source routing the complete hop-by-hop route to be followed by the message is specified by the source, in geographic routing the source simply addresses the message with the position of the destination. As the destination can generally be expected to move slowly compared to the frequency of topology changes between the source and the destination, it makes sense to keep track of the position of the destination instead of maintaining network topology information up to date; if the destination does not move too fast, the message is delivered regardless of possible topology changes among intermediate nodes.

The cost bounds presented in this entry are achieved on *unit disk graphs*. A unit disk graph is defined as follows:

Definition 2 (Unit Disk Graph) Let $V \subset \mathbb{R}^2$ be a set of points in the 2-dimensional plane. The graph with edges between all nodes with distance at most 1 is called the unit disk graph of V.

Unit disk graphs are often employed to model wireless ad hoc networks.

The routing algorithms considered in this entry operate on planar graphs, graphs that contain no two intersecting edges. There exist strictly local algorithms constructing such planar graphs given a unit disk graph. The edges of planar graphs partition the Euclidean plane into contiguous areas, so-called faces. The algorithms cited in this entry are based on these faces.

Key Results

The first geographic routing algorithm shown to always reach the destination was Face Routing introduced in [14].

Theorem 1 *If the source and the destination are connected, Face Routing executed on an arbitrary planar graph always finds a path to the destination. It thereby takes at most $O(n)$ steps, where n is the total number of nodes in the network.*

There exists however a geographic routing algorithm whose cost is bounded not only with respect to the total number of nodes, but in relation to the *shortest path* between the source and the destination: The GOAFR$^+$ algorithm [15,16,18,24] (pronounced as "gopher-plus") combines *greedy routing*—where every intermediate node relays the message to be routed to its neighbor located nearest to the destination—with face routing. Together with the locally computable *Gabriel Graph* planarization technique, the effort expended by the GOAFR$^+$ algorithm is bounded as follows:

Theorem 2 *Let c be the cost of an optimal path from s to t in a given unit disk graph. GOAFR$^+$ reaches t with cost $O(c^2)$ if s and t are connected. If s and t are not connected, GOAFR$^+$ reports so to the source.*

On the other hand it can be shown that—on certain worst-case graphs—no geographic routing algorithm operating in compliance with the above definition can perform asymptotically better than GOAFR$^+$:

Theorem 3 *There exist graphs where any deterministic (randomized) geographic ad hoc routing algorithm has (expected) cost $\Omega(c^2)$.*

This leads to the following conclusion:

Theorem 4 *The cost expended by GOAFR$^+$ to reach the destination on a unit disk graph is asymptotically optimal.*

In addition, it has been shown that the GOAFR$^+$ algorithm is not only guaranteed to have low worst-case cost but that it also performs well in average-case networks with nodes randomly placed in the plane [15,24].

Applications

By its strictly local nature geographic routing is particularly well suited for application in potentially highly dynamic wireless ad hoc networks. However, also its employment in dynamic networks in general is conceivable.

Open Problems

A number of problems related to geographic routing remain open. This is true above all with respect to the dissemination within the network of information about the destination position and on the other hand in the context of node mobility as well as network dynamics. Various approaches to these problems have been described in [7] as well as in chapters 11 and 12 of [24]. More generally, taking geographic routing one step further towards its application in practical wireless ad hoc networks [12,13] is a field yet largely open. A more specific open problem is finally posed by the question whether geographic routing can be adapted to networks with nodes embedded in three-dimensional space.

Experimental Results

First experiences with geographic and in particular face routing in practical networks have been made [12,13]. More specifically, problems in connection with graph planarization that can occur in practice were observed, documented, and tackled.

Cross References

▶ Local Computation in Unstructured Radio Networks
▶ Planar Geometric Spanners
▶ Routing in Geometric Networks

Recommended Reading

1. Barrière, L., Fraigniaud, P., Narayanan, L.: Robust Position-Based Routing in Wireless Ad Hoc Networks with Unstable Transmission Ranges. In: Proc. of the 5th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DIAL-M), pp 19–27. ACM Press, New York (2001)
2. Bose, P., Brodnik, A., Carlsson, S., Demaine, E., Fleischer R., López-Ortiz, A., Morin, P., Munro, J.: Online Routing in Convex Subdivisions. In: International Symposium on Algorithms and Computation (ISAAC). LNCS, vol. 1969, pp 47–59. Springer, Berlin/New York (2000)
3. Bose, P., Morin, P.: Online Routing in Triangulations. In: Proc. 10th Int. Symposium on Algorithms and Computation (ISAAC). LNCS, vol. 1741, pp 113–122. Springer, Berlin (1999)
4. Bose, P.,Morin, P., Stojmenovic, I., Urrutia J.: Routing with Guaranteed Delivery in Ad Hoc Wireless Networks. In: Proc. of the 3rd International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DIAL-M), 1999, pp 48–55
5. Datta, S., Stojmenovic, I., Wu J.: Internal Node and Shortcut Based Routing with Guaranteed Delivery in Wireless Networks. In: Cluster Computing 5, pp 169–178. Kluwer Academic Publishers, Dordrecht (2002)
6. Finn G.: Routing and Addressing Problems in Large Metropolitan-scale Internetworks. Tech. Report ISI/RR-87–180, USC/ISI, March (1987)
7. Flury, R., Wattenhofer, R.: MLS: An Efficient Location Service for Mobile Ad Hoc Networks. In: Proceedings of the 7th ACM Int. Symposium on Mobile Ad-Hoc Networking and Computing (MobiHoc), Florence, Italy, May 2006
8. Fonseca, R., Ratnasamy, S., Zhao, J., Ee, C.T., Culler, D., Shenker, S., Stoica, I.: Beacon Vector Routing: Scalable Point-to-Point Routing in Wireless Sensornets. In: 2nd Symposium on Networked Systems Design & Implementation (NSDI), Boston, Massachusetts, USA, May 2005
9. Gao, J., Guibas, L., Hershberger, J., Zhang, L., Zhu, A.: Geometric Spanner for Routing in Mobile Networks. In: Proc. 2nd ACM Int. Symposium on Mobile Ad-Hoc Networking and Computing (MobiHoc), Long Beach, CA, USA, October 2001
10. Hou, T., Li, V.: Transmission Range Control in Multihop Packet Radio Networks. IEEE Tran. Commun. **34**, 38–44 (1986)
11. Karp, B., Kung, H.: GPSR: Greedy Perimeter Stateless Routing for Wireless Networks. In: Proc. 6th Annual Int. Conf. on Mobile Computing and Networking (MobiCom), 2000, pp 243–254
12. Kim, Y.J., Govindan, R., Karp, B., Shenker, S.: Geographic Routing Made Practical. In: Proceedings of the Second USENIX/ACM Symposium on Networked System Design and Implementation (NSDI 2005), Boston, Massachusetts, USA, May 2005
13. Kim, Y.J., Govindan, R., Karp, B., Shenker, S.: On the Pitfalls of Geographic Face Routing. In: Proc. of the ACM Joint Workshop on Foundations of Mobile Computing (DIALM-POMC), Cologne, Germany, September 2005
14. Kranakis, E., Singh, H., Urrutia, J.: Compass Routing on Geometric Networks. In: Proc. 11th Canadian Conference on Computational Geometry, Vancouver, August 1999, pp 51–54
15. Kuhn, F., Wattenhofer, R., Zhang, Y., Zollinger, A.: Geometric Routing: Of Theory and Practice. In: Proc. of the 22nd ACM Symposium on the Principles of Distributed Computing (PODC), 2003
16. Kuhn, F., Wattenhofer, R., Zollinger, A.: Asymptotically Optimal Geometric Mobile Ad-Hoc Routing. In: Proc. 6th Int. Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (Dial-M), pp 24–33. ACM Press, New York (2002)
17. Kuhn, F., Wattenhofer, R., Zollinger, A.: Ad-Hoc Networks Beyond Unit Disk Graphs. In: 1st ACM Joint Workshop on Foundations of Mobile Computing (DIALM-POMC), San Diego, California, USA, September 2003
18. Kuhn, F., Wattenhofer, R., Zollinger, A.: Worst-Case Optimal and Average-Case Efficient Geometric Ad-Hoc Routing. In: Proc. 4th ACM Int. Symposium on Mobile Ad-Hoc Networking and Computing (MobiHoc), 2003

19. Leong, B., Liskov, B., Morris, R.: Geographic Routing without Planarization. In: 3rd Symposium on Networked Systems Design & Implementation (NSDI), San Jose, California, USA, May 2006
20. Leong, B., Mitra, S., Liskov, B.: Path Vector Face Routing: Geographic Routing with Local Face Information. In: 13th IEEE International Conference on Network Protocols (ICNP), Boston, Massachusetts, USA, November 2005
21. Takagi, H., Kleinrock, L.: Optimal Transmission Ranges for Randomly Distributed Packet Radio Terminals. IEEE Trans. Commun. **32**, 246–257 (1984)
22. Urrutia, J.: Routing with Guaranteed Delivery in Geometric and Wireless Networks. In: Stojmenovic, I. (ed.) Handbook of Wireless Networks and Mobile Computing, ch. 18 pp. 393–406. Wiley, Hoboken (2002)
23. Wattenhofer, M., Wattenhofer, R., Widmayer, P.: Geometric Routing without Geometry. In: 12th Colloquium on Structural Information and Communication Complexity (SIROCCO), Le Mont Saint-Michel, France, May 2005
24. Zollinger, A.: Networking Unleashed: Geographic Routing and Topology Control in Ad Hoc and Sensor Networks, Ph. D. thesis, ETH Zurich, Switzerland Diss. ETH 16025 (2005)

Geometric Computing

- ► Engineering Geometric Algorithms
- ► Euclidean Traveling Salesperson Problem
- ► Geographic Routing
- ► Minimum k-Connected Geometric Networks
- ► Planar Geometric Spanners
- ► Point Pattern Matching
- ► Routing in Geometric Networks

Geometric Dilation of Geometric Networks

2006; Dumitrescu, Ebbers-Baumann, Grüne, Klein, Knauer, Rote

ROLF KLEIN
Institute for Computer Science I, University of Bonn, Bonn, Germany

Keywords and Synonyms

Detour; Spanning ratio; Stretch factor

Problem Definition

Urban street systems can be modeled by *plane geometric networks* $G = (V, E)$ whose edges $e \in E$ are piecewise smooth curves that connect the vertices $v \in V \subset \mathbb{R}^2$. Edges do not intersect, except at common endpoints in V. Since streets are lined with houses, the quality of such a network can be measured by the length of the connections it provides between two arbitrary points p and q on G.

Let $\xi_G(p, q)$ denote a shortest path from p to q in G. Then

$$\delta(p, q) := \frac{|\xi_G(p, q)|}{|pq|} \tag{1}$$

is the detour one encounters when using network G, in order to get from p to q, instead of walking straight. Here, $|.|$ denotes the Euclidean length. The *geometric dilation of network* G is defined by

$$\delta(G) := \sup_{p \neq q \in G} \delta(p, q). \tag{2}$$

This definition differs from the notion of stretch factor (or: spanning ratio) used in the context of spanners; see the monographs by Eppstein [6] or Narasimhan and Smid [11]. In the latter, only the paths between the vertices $p, q \in V$ are considered, whereas the geometric dilation involves all points on the edges as well. As a consequence, the stretch factor of a triangle T equals 1, but its geometric dilation is given by $\delta(T) = \sqrt{2/(1 - \cos\alpha)} \geq 2$, where $\alpha \leq 60°$ is the most acute angle of T.

Presented with a finite set S of points in the plane, one would like to find a finite geometric network containing S whose geometric dilation is as small as possible. The value of

$$\Delta(S) := \inf\{\delta(G); G \text{ finite plane geometric network containing } S\}$$

is called the *geometric dilation of point set* S. The problem is in computing, or bounding, $\Delta(S)$ for a given set S.

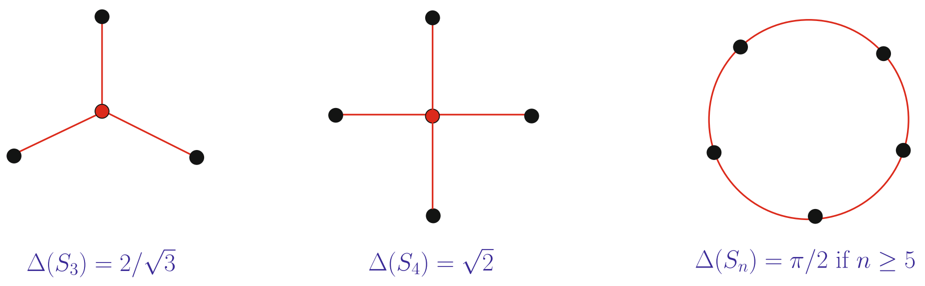
Key Results

Theorem 1 [4] *Let S_n denote the set of corners of a regular n-gon. Then, $\Delta(S_3) = 2/\sqrt{3}$, $\Delta(S_4) = \sqrt{2}$, and $\Delta(S_n) = \pi/2$ for all $n \geq 5$.*

The networks realizing these minimum values are shown in Fig. 1. The proof of minimality uses the following two lemmata that may be interesting in their own right. Lemma 1 was independently obtained by Aronov et al. [1].

Lemma 1 *Let T be a tree containing S_n. Then $\delta(T) \geq n/\pi$.*

Lemma 2 follows from a result of Gromov's [7]. It can more easily be proven by applying Cauchy's surface area formula, see [4].


$\Delta(S_3) = 2/\sqrt{3}$
$\Delta(S_4) = \sqrt{2}$
$\Delta(S_n) = \pi/2$ if $n \geq 5$

Geometric Dilation of Geometric Networks, Figure 1
Minimum dilation embeddings of regular point sets

Lemma 2 *Let C denote a simple closed curve in the plane. Then $\delta(C) \geq \pi/2$.*

Clearly, Lemma 2 is tight for the circle. The next lemma implies that the circle is the only closed curve attaining the minimum geometric dilation of $\pi/2$.

Lemma 3 [3] *Let C be a simple closed curve of geometric dilation $< \pi/2 + \epsilon(\delta)$. Then C is contained in an annulus of width δ.*

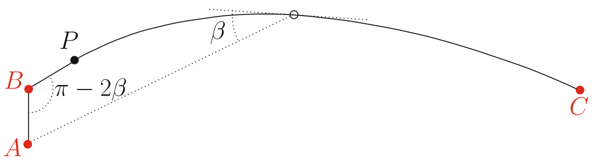
For points in general position, computing their geometric dilation seems quite complicated. Only for sets $S = \{A, B, C\}$ of size three is the solution completely known.

Theorem 2 [5] *The plane geometric network of minimum geometric dilation containing three given points $\{A, B, C\}$ is either a line segment, or a Steiner tree as depicted in Fig. 1, or a simple path consisting of two line segments and one segment of an exponential spiral; see Fig. 2.*

The optimum path shown in Fig. 2 contains a degree two Steiner vertex, P, situated at distance $|AB|$ from B. The path runs straight between A, B and B, P. From P to C it follows an exponential spiral centered at A.

The next results provide upper and lower bounds to $\Delta(S)$.

Theorem 3 [4] *For each finite point set S the estimate $\Delta(S) < 1.678$ holds.*

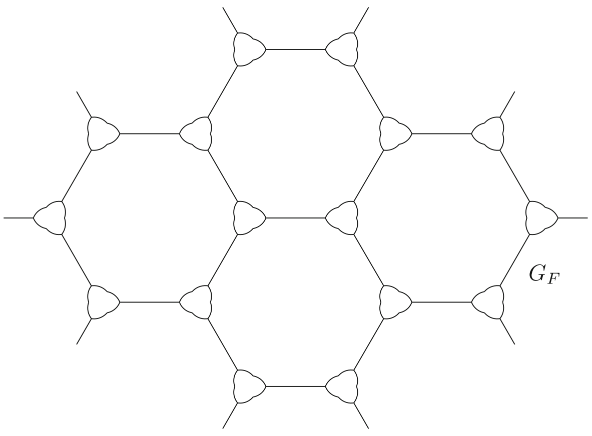


Geometric Dilation of Geometric Networks, Figure 2
The minimum dilation embedding of points A, B, and C

To prove this general upper bound one can replace each vertex of the hexagonal tiling of $\mathbb{R}^2$ with a certain closed Zindler curve (by definition, all point pairs bisecting the perimeter of a Zindler curve have identical distance). This results in a network G_F of geometric dilation ≈ 1.6778; see Fig. 3. Given a finite point set S, one applies a slight deformation to a scaled version of G_F, such that all points of S lie on a finite part, G, of the deformed net. By Dirichlet's result on simultaneous approximation of real numbers by rationals, a deformation small as compared to the cell size is sufficient, so that the dilation is not affected. See [8] for the history and properties of Zindler curves.

Theorem 4 [3] *There exists a finite point set S such that $\Delta(S) > (1 + 10^{-11})\pi/2$.*

Theorem 4 holds for the set S of 19×19 vertices of the integer grid. Roughly, if S were contained in a geometric network G of dilation close to $\pi/2$, the boundaries of the faces of G must be contained in small annuli, by Lemma 3. To the inner and outer circles of these annuli, one can now apply a result by Kuperberg et al. [9] stating that an enlarge-



Geometric Dilation of Geometric Networks, Figure 3
A network of geometric dilation ≈ 1,6778

ment, by a certain factor, of a packing of disks of radius ≤ 1 cannot cover a square of size 4.

Applications

The geometric dilation has applications in the theory of knots, see, e. g., Kusner and Sullivan [10] and Denne and Sullivan [2]. With respect to urban planning, the above results highlight principal dilation bounds for connecting given sites with plane geometric networks.

Open Problems

For practical applications, one would welcome upper bounds to the weight (= total edge length) of a geometric network, in addition to upper bounds on its geometric dilation. Some theoretical questions require further investigation, too. Is $\Delta(S)$ always attained by a finite network? How to compute, or approximate, $\Delta(S)$ for a given finite set S? What is the precise value of $\sup\{\Delta(S); S \text{ finite}\}$?

Cross References

► Dilation of Geometric Networks

Recommended Reading

1. Aronov, B., de Berg, M., Cheong, O., Gudmundsson, J., Haverkort, H., Vigneron, A.: Sparse Geometric Graphs with Small Dilation. 16th International Symposium ISAAC 2005, Sanya. In: Deng, X., Du, D. (eds.) Algorithms and Computation, Proceedings. LNCS, vol. 3827, pp. 50–59. Springer, Berlin (2005)
2. Denne, E., Sullivan, J.M.: The Distortion of a Knotted Curve. http://www.arxiv.org/abs/math.GT/0409438 (2004)
3. Dumitrescu, A., Ebbers-Baumann, A., Grüne, A., Klein, R., Rote, G.: On the Geometric Dilation of Closed Curves, Graphs, and Point Sets. Comput. Geom. Theory Appl. **36**(1), 16–38 (2006)
4. Ebbers-Baumann, A., Grüne, A., Klein, R.: On the Geometric Dilation of Finite Point Sets. Algorithmica **44**(2), 137–149 (2006)
5. Ebbers-Baumann, A., Klein, R., Knauer, C., Rote, G.: The Geometric Dilation of Three Points. Manuscript (2006)
6. Eppstein, D.: Spanning Trees and Spanners. In: Sack, J.-R., Urrutia, J. (eds.) Handbook of Computational Geometry, pp. 425–461. Elsevier, Amsterdam (1999)
7. Gromov, M.: Structures Métriques des Variétés Riemanniennes. Textes Math. CEDIX, vol. 1. F. Nathan, Paris (1981)
8. Grüne, A.: Geometric Dilation and Halving Distance. Ph. D. thesis, Institut für Informatik I, Universität Bonn (2006)
9. Kuperberg, K., Kuperberg, W., Matousek, J., Valtr, P.: Almost Tiling the Plane with Ellipses. Discrete Comput. Geom. **22**(3), 367–375 (1999)
10. Kusner, R.B., Sullivan, J.M.: On Distortion and Thickness of Knots. In: Whittington, S.G. et al. (eds.) Topology and Geometry in Polymer Science. IMA Volumes in Math. and its Applications, vol. 103, pp. 67–78. Springer, New York (1998)
11. Narasimhan, G., Smid, M.: Geometric Spanner Networks. Cambridge University Press(2007)

Geometric Spanners

2002; Gudmundsson, Levcopoulos, Narasimhan

JOACHIM GUDMUNDSSON[1], GIRI NARASIMHAN[2], MICHIEL SMID[3]

[1] DMiST, National ICT Australia Ltd, Alexandria, NSW, Australia

[2] Department of Computer Science, Florida International University, Miami, FL, USA

[3] School of Computer Science, Carleton University, Ottawa, ON, Canada

Keywords and Synonyms

Dilation; t-spanners

Problem Definition

Consider a set S of n points in d-dimensional Euclidean space. A network on S can be modeled as an undirected graph G with vertex set S of size n and an edge set E where every edge (u, v) has a weight. A geometric (Euclidean) network is a network where the weight of the edge (u, v) is the Euclidean distance $|uv|$ between its endpoints. Given a real number $t > 1$ we say that G is a *t-spanner* for S, if for each pair of points $u, v \in S$, there exists a path in G of weight at most t times the Euclidean distance between u and v. The minimum t such that G is a t-spanner for S is called the stretch factor, or dilation, of G. For a more detailed description of the construction of t-spanners see the book by Narasimhan and Smid [18]. The problem considered is the construction of t-spanners given a set S of n points in $\mathcal{R}^d$ and a positive real value $t > 1$, where d is a constant. The aim is to compute a good t-spanner for S with respect to the following quality measures:

size: the number of edges in the graph.

degree: the maximum number of edges incident on a vertex.

weight: the sum of the edge weights.

spanner diameter: the smallest integer k such that for any pair of vertices u and v in S, there is a path in the graph of length at most $t \cdot |uv|$ between u and v containing at most k edges.

fault-tolerance: the resilience of the graph to edge, vertex or region failures.

Thus, good t-spanners require large fault-tolerance and small size, degree, weight and spanner diameter.

Key Results

This section contains a description of the three most common approaches for constructing a t-spanner of a set of points in Euclidean space. It also contains a description of the construction of fault-tolerant spanners, spanners among polygonal obstacles and, finally, a short note on dynamic and kinetic spanners.

Spanners of Points in Euclidean Space

The most well-known classes of t-spanner networks for points in Euclidean space include: Θ-graphs, WSPD-graphs and Greedy-spanners. In the following sections the main idea of each of these classes is given, together with the known bounds on the quality measures.

The $\boldsymbol{\Theta}$-Graph The Θ-graph was discovered independently by Clarkson and Keil in the late 80's. The general idea is to process each point $p \in S$ independently as follows. Partition $\mathcal{R}^d$ into k simplicial cones of angular diameter at most θ and apex at p, where $k = O(1/\theta^{d-1})$. For each non-empty cone C, an edge is added between p and the point in C whose orthogonal projection onto some fixed ray in C emanating from p is closest to p, see Fig. 1a. The resulting graph is called the Θ-graph on S.

Theorem 1 *The Θ-graph is a t-spanner of S for $t = 1/(\cos\theta - \sin\theta)$ with $O(n/\theta^{d-1})$ edges, and can be computed in $O((n/\theta^{d-1})\log^{d-1} n)$ time using $O(n/\theta^{d-1} + n\log^{d-2} n)$ space.*

The following variants of the Θ-graph also give bounds on the degree, diameter and weight.

Skip-List Spanners The idea is to generalize skip-lists and apply them to the construction of spanners. Construct a sequence of h subsets, $S_1, \ldots, S_h$, where $S_1 = S$ and S_i is constructed from S_{i-1} as follows (reminiscent of the levels in a skip list). For each point in S_{i-1}, flip a fair coin. The set S_i is the set of all points of S_{i-1} whose coin flip produced heads. The construction stops if $S_i = \emptyset$. For each subset a Θ-graph is constructed. The union of the graphs is the skip-list spanner of S with dilation t, having $O(n/\theta^{d-1})$ edges and $O(\log n)$ spanner diameter with high probability [3].

Gap-Greedy A set of directed edges is said to satisfy the *gap* property if the sources of any two distinct edges in the set are separated by a distance that is at least proportional to the length of the shorter of the two edges. Arya and Smid [5] proposed an algorithm that uses the gap property to decide whether or not an edge should be added to the t-spanner graph. Using the gap property the constructed spanner can be shown to have degree $O(1/\theta^{d-1})$ and weight $O(\log n \cdot wt(\text{MST}(S)))$, where $wt(\text{MST}(S))$ is the weight of the minimum spanning tree of S.

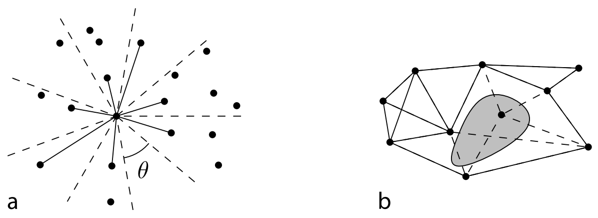
The WSPD-Graph Let A and B be two finite sets of points in $\mathcal{R}^d$. We say that A and B are *well-separated with respect to* a real value $s > 0$, if there are two disjoint balls C_A and C_B, having the same radius, such that C_A contains A, C_B contains B, and the distance between C_A and C_B is at least equal to s times the radius of C_A. The value s is denoted the *separation ratio*.

Definition 1 ([6]) Let S be a set of points in $\mathcal{R}^d$, and let $s > 0$ be a real number. A *well-separated pair decomposition* (WSPD) for S with respect to s is a sequence $\{A_i, B_i\}$, $1 \leq i \leq m$, of pairs of non-empty subsets of S, such that (1) $A_i \cap B_i = \emptyset$ for all $i = 1, 2, \ldots, m$, (2) for each unordered pair $\{p, q\}$ of distinct points of S, there is exactly one pair $\{A_i, B_i\}$ in the sequence, such that $p \in A_i$ and $q \in B_i$, or $p \in B_i$ and $q \in A_i$, and (3) A_i and B_i are well-separated with respect to s, for all $i = 1, 2, \ldots, m$.

The well-separated pair decomposition (WSPD) was developed by Callahan and Kosaraju [6]. The construction of a t-spanner using the well-separated pair decomposition is done by first constructing a WSPD of S with respect to a separation constant $s = (4(t+1))/(t-1)$. Initially set the spanner graph $G = (S, \emptyset)$ and add edges iteratively as follows. For each well-separated pair $\{A, B\}$ in the decomposition, an edge (a, b) is added to the graph, where a and b are arbitrary points in A and B, respectively. The resulting graph is called the WSPD-graph on S.

Theorem 2 *The WSPD-graph is a t-spanner for S with $O(s^d \cdot n)$ edges and can be constructed in time $O(s^d n + n\log n)$, where $s = 4(t+1)/(t-1)$.*

There are modifications that can be made to obtain bounded diameter or bounded degree.



Geometric Spanners, Figure 1
a Illustrating the $\boldsymbol{\Theta}$-graph. b A graph with a region-fault

Bounded Diameter Arya, Mount and Smid [3] showed how to modify the construction algorithm such that the diameter of the graph is bounded by $2 \log n$. Instead of selecting an arbitrary point in each well-separated set, their algorithm carefully chooses a specially selected point for each set.

Bounded Degree A single point v can be part of many well-separated pairs and each of these pairs may generate an edge with an endpoint at v. Arya et al. [2] suggested an algorithm that retains only the shortest edge for each cone direction, thus combining the Θ-graph approach with the WSPD-graph. By adding a post-processing step that handles all high-degree vertices, a t-spanner of degree $O(1/(t-1)^{2d-1})$ is obtained.

The Greedy-Spanner The greedy algorithm was first presented in 1989 by Bern (see also Levcopoulos and Lingas [15]) and since then the greedy algorithm has been subject to considerable research. The graph constructed using the greedy algorithm is called a Greedy-spanner, and the general idea is that the algorithm iteratively builds a graph G. The edges in the complete graph are processed in order of increasing edge length. Testing an edge (u, v) entails a shortest path query in the partial spanner graph G. If the shortest path in G between u and v is at most $t \cdot |uv|$ then the edge (u, v) is discarded, otherwise it is added to the partial spanner graph G.

Das, Narasimhan and Salowe [11] proved that the greedy-spanner fulfills the so-called *leapfrog property*. A set of undirected edges E is said to satisfy the t-leapfrog property, if for every $k \geq 2$, and for every possible sequence $\{(p_1, q_1), \ldots, (p_k, q_k)\}$ of pairwise distinct edges of E,

$$t \cdot |p_1 q_1| < \sum_{i=2}^{k} |p_i q_i| + t \cdot \left(\sum_{i=1}^{k-1} |q_i p_{i+1}| + |p_k q_1| \right) .$$

Using the leapfrog property it is possible to bound the weight of the graph. Das and Narasimhan [10] observed that the Greedy-spanner can be approximated while maintaining the leapfrog property. This observation allowed for faster construction algorithms.

Theorem 3 **([14])** *The approximate greedy-spanner is a t-spanner of S with maximum degree $O(1/(t-1)^{2d-1})$, weight $O((1/(t-1)^{2d-1} \cdot wt(MST(S))))$, and can be computed in time $O(n/((t-1)^{2d}) \log n)$.*

Fault-Tolerant Spanners

The concept of fault-tolerant spanners was first introduced by Levcopoulos et al. [16] in 1998, i. e., after one or more vertices or edges fail, the spanner should retain its good properties. In particular, there should still be a short path between any two vertices in what remains of the spanner after the fault. Czumaj and Zhao [8] showed that a greedy approach produces a k-vertex (or k-edge) fault tolerant geometric t-spanner with degree $O(k)$ and total weight $O(k^2 \cdot wt(MST(S)))$; these bounds are asymptotically optimal.

For geometric spanners it is natural to consider *region faults*, i. e., faults that destroy all vertices and edges intersecting some geometric fault region. For a fault region F let $G \ominus F$ be the part of G that remains after the points from S inside F and all edges that intersect F have been removed from the graph, see Fig. 1b. Abam et al. [1] showed how to construct region-fault tolerant t-spanners of size $O(n \log n)$ that are fault-tolerant to any convex region-fault. If one is allowed to use Steiner points then a linear size t-spanner can be achieved.

Spanners Among Obstacles

The visibility graph of a set of pairwise non-intersecting polygons is a graph of intervisible locations. Each polygonal vertex is a vertex in the graph, and each edge represents a visible connection between them; that is, if two vertices can see each other, an edge is drawn between them. This graph is useful since it contains the shortest obstacle avoiding path between any pair of vertices.

Das [9] showed that a t-spanner of the visibility graph of a point set in the Euclidean plane can be constructed by using the Θ-graph approach followed by a pruning step. The obtained graph has linear size and constant degree.

Dynamic and Kinetic Spanners

Not much is known in the areas of dynamic or kinetic spanners. Arya et al. [4] showed a data structure of size $O(n \log^d n)$ that maintains the skip-list spanner, described in Sect. "The Θ-Graph", in $O(\log^d n \log \log n)$ expected amortized time per insertion and deletion in the model of random updates.

Gao et al. [13] showed how to maintain a t-spanner of size $O(n/(t-1)^d)$ and maximum degree $O(1/(t-2)^d \log \alpha)$ in time $O((\log \alpha)/(t-1)^d)$ per insertion and deletion, where α denotes the aspect ratio of S, i. e., the ratio of the maximum pairwise distance to the minimum pairwise distance. The idea is to use an hierarchical structure T with $O(\log \alpha)$ levels, where each level contains a set of centers (subset of S). Each vertex v on level i in T is connected by an edge to all other vertices on level i within distance $O(2^i/(t-1))$ of v. The resulting graph is a t-spanner of S and it can be maintained as stated above. The approach can be generalized to the kinetic case so that the total number of events in maintaining the spanner is $O(n^2 \log n)$ under pseudo-algebraic motion. Each event can be updated in $O((\log \alpha)/(t-1)^d)$ time.

Applications

The construction of sparse spanners has been shown to have numerous applications areas such as metric space searching [1], which includes query by content in multimedia objects, text retrieval, pattern recognition and function approximation. Another example is broadcasting in communication networks [17]. Several well-known theoretical results also use the construction of t-spanners as a building block, for example, Rao and Smith [19] made a breakthrough by showing an optimal $O(n \log n)$-time approximation scheme for the well-known Euclidean *traveling salesperson problem*, using t-spanners (or banyans). Similarly, Czumaj and Lingas [7] showed approximation schemes for minimum-cost multi-connectivity problems in geometric networks.

Open Problems

There are many open problems in this area. Only a few are mentioned here:

1. Design a dynamic t-spanner that can be updated in $O(\log^c n)$ time, for some constant c.
2. Determine if there exists a fault-tolerant t-spanner of linear size for convex region faults.
3. The k-vertex fault tolerant spanner by Czumaj and Zhao [8] produces a k-vertex fault tolerant t-spanner of degree $O(k)$ and weight $O(k^2 \cdot wt(MST(S)))$. However, it is not known how to implement it efficiently. Can such a spanner be computed in $O(n \log n + kn)$ time?
4. Bound the weight of skip-list spanners.

Experimental Results

The problem of constructing spanners has received considerable attention from a theoretical perspective but not much attention from a practical, or experimental perspective. Navarro and Paredes [1] presented four heuristics for point sets in high-dimensional space ($d = 20$) and showed by empirical methods that the running time was $O(n^{2.24})$ and the number of edges in the produced graphs was $O(n^{1.13})$. Recently Farshi and Gudmundsson [12] performed a thorough comparison of the construction algorithms discussed in Section "Spanners of Points in Euclidean Space".

Cross References

- ▶ Applications of Geometric Spanner Networks
- ▶ Approximating Metric Spaces by Tree Metrics
- ▶ Dilation of Geometric Networks
- ▶ Planar Geometric Spanners
- ▶ Single-Source Shortest Paths
- ▶ Sparse Graph Spanners
- ▶ Well Separated Pair Decomposition

Recommended Reading

1. Abam, M.A., de Berg, M., Farshi, M., Gudmundsson, J.: Region-fault tolerant geometric spanners. In: Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms, New Orleans, 7–9 January 2007
2. Arya, S., Das, G., Mount, D.M., Salowe, J.S., Smid, M.: Euclidean spanners: short, thin, and lanky. In: Proceedings of the 27th ACM Symposium on Theory of Computing, pp. 489–498. Las Vegas, 29 May–1 June 1995
3. Arya, S., Mount, D.M., Smid, M.: Randomized and deterministic algorithms for geometric spanners of small diameter. In: Proceedings of the 35th IEEE Symposium on Foundations of Computer Science, pp. 703–712. Santa Fe, 20–22 November 1994
4. Arya, S., Mount, D.M., Smid, M.: Dynamic algorithms for geometric spanners of small diameter: Randomized solutions. Comput. Geom. Theor. Appl. **13**(2), 91–107 (1999)
5. Arya, S., Smid, M.: Efficient construction of a bounded-degree spanner with low weight. Algorithmica **17**, 33–54 (1997)
6. Callahan, P.B., Kosaraju, S.R.: A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. J. ACM **42**, 67–90 (1995)
7. Czumaj, A., Lingas, A.: Fast approximation schemes for Euclidean multi-connectivity problems. In: Proceedings of the 27th International Colloquium on Automata, Languages and Programming. Lect. Notes Comput. Sci. **1853**, 856–868 (2000)
8. Czumaj, A., Zhao, H.: Fault-tolerant geometric spanners. Discret. Comput. Geom. **32**(2), 207–230 (2004)
9. Das, G.: The visibility graph contains a bounded-degree spanner. In: Proceedings of the 9th Canadian Conference on Computational Geometry, Kingston, 11–14 August 1997
10. Das, G., Narasimhan, G.: A fast algorithm for constructing sparse Euclidean spanners. Int. J. Comput. Geom. Appl. **7**, 297–315 (1997)
11. Das, G., Narasimhan, G., Salowe, J.: A new way to weigh malnourished Euclidean graphs. In: Proceedings of the 6th ACM-SIAM Symposium on Discrete Algorithms, pp. 215–222. San Francisco, 22–24 January 1995
12. Farshi, M., Gudmundsson, J.: Experimental study of geometric t-spanners. In: Proceedings of the 13th Annual European Symposium on Algorithms. Lect. Notes Comput. Sci. **3669**, 556–567 (2005)
13. Gao, J., Guibas, L.J., Nguyen, A.: Deformable spanners and applications. In: Proceedings of the 20th ACM Symposium on Computational Geometry, pp. 190–199, New York, 9–11 June 2004

14. Gudmundsson, J., Levcopoulos, C., Narasimhan, G.: Improved greedy algorithms for constructing sparse geometric spanners. SIAM J. Comput. **31**(5), 1479–1500 (2002)
15. Levcopoulos, C., Lingas, A.: There are planar graphs almost as good as the complete graphs and almost as cheap as minimum spanning trees. Algorithmica **8**(3), 251–256 (1992)
16. Levcopoulos, C., Narasimhan, G., Smid, M.: Improved algorithms for constructing fault-tolerant spanners. Algorithmica **32**, 144–156 (2002)
17. Li, X.Y.: Applications of computational geometry in wireless ad hoc networks. In: Cheng, X.Z., Huang, X., Du, D.Z. (eds.) Ad Hoc Wireless Networking, pp. 197–264. Kluwer, Dordrecht (2003)
18. Narasimhan, G., Smid, M.: Geometric spanner networks. Cambridge University Press, New York (2006)
19. Navarro, G., Paredes, R.: Practical construction of metric t-spanners. In: Proceedings of the 5th Workshop on Algorithm Engineering and Experiments, pp. 69–81, 11 January 2003. SIAM Press, Baltimore
20. Rao, S., Smith, W.D.: Approximating geometrical graphs via spanners and banyans. In: Proceedings of the 30th ACM Symposium on Theory of Computing, pp. 540–550. Dallas, 23–26 May 1998

Gomory–Hu Trees

2007; Bhalgat, Hariharan, Kavitha, Panigrahi

Debmalya Panigrahi
Computer Science & Artificial Intelligence Laboratory, MIT, Cambridge, MA, USA

Keywords and Synonyms

Cut trees

Problem Definition

Let $G = (V, E)$ be an undirected graph with $|V| = n$ and $|E| = m$. The edge connectivity of two vertices $s, t \in V$, denoted by $\lambda(s, t)$, is defined as the size of the smallest cut that separates s and t; such a cut is called a minimum s–t cut. Clearly, one can represent the $\lambda(s, t)$ values for all pairs of vertices s and t in a table of size $O(n^2)$. However, for reasons of efficiency, one would like to represent all the $\lambda(s, t)$ values in a more succinct manner. *Gomory–Hu trees* (also known as *cut trees*) offer one such succinct representation of linear (i. e., $O(n)$) space and constant (i. e., $O(1)$) lookup time. It has the additional advantage that apart from representing all the $\lambda(s, t)$ values, it also contains structural information from which a minimum s–t cut can be retrieved easily for any pair of vertices s and t.

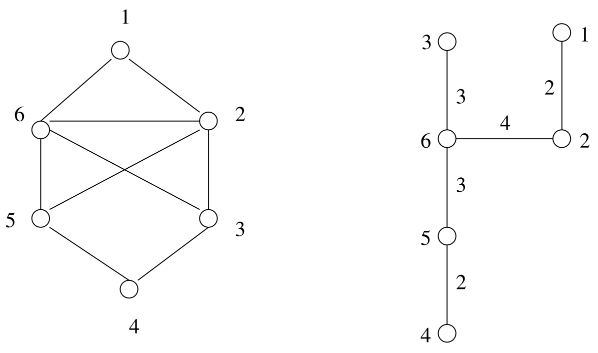
Formally, a Gomory–Hu tree $T = (V, F)$ of an undirected graph $G = (V, E)$ is a weighted undirected tree defined on the vertices of the graph such that the following properties are satisfied:

- For any pair of vertices $s, t \in V$, $\lambda(s, t)$ is equal to the minimum weight on an edge in the unique path connecting s to t in T. Call this edge $e(s, t)$. If there are multiple edges with the minimum weight on the s to t path in T, any one of these edges is designated as $e(s, t)$.
- For any pair of vertices s and t, the bipartition of vertices into components produced by removing $e(s, t)$ (if there are multiple candidates for $e(s, t)$, this property holds for each candidate edge) from T corresponds to a minimum s–t cut in the original graph G.

To understand this definition better, consider the following example. Figure 1 shows an undirected graph and a corresponding Gomory–Hu tree. Focus on a pair of vertices, for instance, 3 and 5. Clearly, the edge (6, 5) of weight 3 is a minimum-weight edge on the 3 to 5 path in the Gomory–Hu tree. It is easy to see that $\lambda(3, 5) = 3$ in the original graph. Now, removing this edge produces the vertex bipartition $(\{1, 2, 3, 6\}, \{4, 5\})$, which is a cut of size 3 in the original graph.

It is not immediate that such Gomory–Hu trees exist for all undirected graphs. In a classical result in 1961, Gomory and Hu [7] showed that not only do such trees exist for all undirected graphs, but that they can also be computed using $n - 1$ minimum s–t computations (which are equivalent to maximum flow computations, by the celebrated Menger's theorem). In fact, a graph can have multiple Gomory–Hu trees.

All previous algorithms for building Gomory–Hu trees in undirected graphs used maximum flow subroutines. Gomory and Hu showed how to compute the cut tree T using $n - 1$ maximum flow computations and graph contractions. Gusfield [8] proposed an algorithm that does not use graph contractions; all $n - 1$ maximum flow compu-



Gomory–Hu Trees, Figure 1
An undirected graph (*left*) and a corresponding Gomory–Hu tree (*right*)

tations are performed on the input graph. Goldberg and Tsioutsiouliklis [6] did an experimental study of the algorithms due to Gomory and Hu and due to Gusfield for the cut tree problem and described efficient implementations of these algorithms. Examples were shown by Benczúr [1] that cut trees do not exist for directed graphs.

Any maximum flow based approach for constructing a Gomory–Hu tree would have a running time of $(n-1)$ times the time for computing a single maximum flow. Till now, faster algorithms for Gomory–Hu trees were by-products of faster algorithms for computing a maximum flow. The current fastest $\tilde{O}(m + n\lambda(s,t))$ (polylog n factors ignored in $\tilde{O}$ notation) maximum-flow algorithm, due to Karger and Levine [10], yields the current best expected running time of $\tilde{O}(n^3)$ for Gomory–Hu tree construction on simple unweighted graphs with n vertices. Bhalgat et al. [2] improved this time complexity to $\tilde{O}(mn)$. Note that both Karger and Levine's algorithm and Bhalgat et al.'s algorithm are randomized Las Vegas algorithms. The fastest deterministic algorithm for the Gomory–Hu tree construction problem is a by-product of Goldberg and Rao's maximum-flow algorithm [5] and has a running time of $\tilde{O}(nm^{1/2}\min(m, n^{3/2}))$.

Key Results

Bhalgat et al. [2] considered the problem of designing an efficient algorithm for constructing a Gomory–Hu tree on unweighted undirected graphs. The main theorem shown in this paper is the following.

Theorem 1 *Let $G = (V, E)$ be a simple unweighted graph with m edges and n vertices. Then a Gomory–Hu tree for G can be built in expected time $\tilde{O}(mn)$.*

Their algorithm is always faster by a factor of $\tilde{\Omega}(n^{2/9})$ (polylog n factors ignored in $\tilde{\Omega}$ notation) compared to the previous best algorithm.

Instead of using maximum flow subroutines, they use a Steiner connectivity algorithm. The *Steiner connectivity* of a set of vertices S (called the *Steiner set*) in an undirected graph is the minimum size of a cut which splits S into two parts; such a cut is called a *minimum Steiner cut*. Generalizing a tree-packing algorithm given by Gabow [4] for finding the edge connectivity of a graph, Cole and Hariharan [3] gave an algorithm for finding the Steiner connectivity k of a set of vertices in either undirected or directed Eulerian unweighted graphs in $\tilde{O}(mk^2)$ time. (For undirected graphs, their algorithm runs a little faster in time $\tilde{O}(m + nk^3)$.) Bhalgat et al. improved this result and gave the following theorem.

Theorem 2 *In an undirected or directed Eulerian unweighted graph, the Steiner connectivity k of a set of vertices can be determined in time $\tilde{O}(mk)$.*

The algorithm in [3] was used by Hariharan et al. [9] to design an algorithm with expected running time $\tilde{O}(m + nk^3)$ to compute a *partial* Gomory–Hu tree for representing the $\lambda(s,t)$ values for all pairs of vertices s, t that satisfied $\lambda(s,t) \le k$. Replacing the algorithm in [3] by the new algorithm for computing Steiner connectivity yields an algorithm to compute a partial Gomory–Hu tree in expected running time $\tilde{O}(m + nk^2)$. Bhalgat et al. showed that using a more detailed analysis this result can be improved to give the following theorem.

Theorem 3 *The partial Gomory–Hu tree of an undirected unweighted graph to represent all $\lambda(s,t)$ values not exceeding k can be constructed in expected time $\tilde{O}(mk)$.*

Since $\lambda(s,t) < n$ for all s, t vertex pairs in an unweighted (and simple) graph, setting k to n in Theorem 3 implies Theorem 1.

Applications

Gomory–Hu trees have many applications in multiterminal network flows and are an important data structure in graph connectivity literature.

Open Problems

The problem of derandomizing the algorithm due to Bhalgat et al. [2] to produce an $\tilde{O}(mn)$ time deterministic algorithm for constructing Gomory–Hu trees for unweighted undirected graphs remains open. The other main challenge is to extend the results in [2] to weighted graphs.

Experimental Results

Goldberg and Tsioutsiouliklis [6] did an extensive experimental study of the cut tree algorithms due to Gomory and Hu [7] and that due to Gusfield [8]. They showed how to efficiently implement these algorithms and also introduced and evaluated heuristics for speeding up the algorithms. Their general observation was that while Gusfield's algorithm is faster in many situations, Gomory and Hu's algorithm is more robust. For more detailed results of their experiments, refer to [6].

No experimental results are reported for the algorithm due to Bhalgat et al. [2].

Cross References

► Approximate Maximum Flow Construction

Recommended Reading

1. Benczúr, A.A.: Counterexamples for Directed and Node Capacitated Cut-Trees. SIAM J. Comput. **24**(3), 505–510 (1995)
2. Bhalgat, A., Hariharan, R., Kavitha, T., Panigrahi, D.: An $\tilde{O}(mn)$ Gomory-Hu tree construction algorithm for unweighted graphs. In: Proc. of the 39th Annual ACM Symposium on Theory of Computing, San Diego 2007
3. Cole, R., Hariharan, R.: A Fast Algorithm for Computing Steiner Edge Connectivity. In: Proc. of the 35th Annual ACM Symposium on Theory of Computing, San Diego 2003, pp. 167–176
4. Gabow, H.N.: A matroid approach to finding edge connectivity and packing arborescences. J. Comput. Syst. Sci. **50**, 259–273 (1995)
5. Goldberg, A.V., Rao, S.: Beyond the Flow Decomposition Barrier. J. ACM **45**(5), 783–797 (1998)
6. Goldberg, A.V., Tsioutsiouliklis, K.: Cut Tree Algorithms: An Experimental Study. J. Algorithms **38**(1), 51–83 (2001)
7. Gomory, R.E., Hu, T.C.: Multi-terminal network flows. J. Soc. Indust. Appl. Math. **9**(4), 551–570 (1961)
8. Gusfield, D.: Very Simple Methods for All Pairs Network Flow Analysis. SIAM J. Comput. **19**(1), 143–155 (1990)
9. Hariharan, R., Kavitha, T., Panigrahi, D.: Efficient Algorithms for Computing All Low *s-t* Edge Connectivities and Related Problems. In: Proc. of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms, 2007, pp. 127–136
10. Karger, D., Levine, M.: Random Sampling in Residual Graphs. In: Proc. of the 34th Annual ACM Symposium on Theory of Computing 2002, pp. 63–66

Graph Bandwidth

1998; Feige
2000; Feige

James R. Lee
Department of Computer Science and Engineering, University of Washington, Seattle, WA, USA

Keywords and Synonyms

Graph bandwidth; Approximation algorithms; Metric embeddings

Problem Definition

The *graph bandwidth problem* concerns producing a linear ordering of the vertices of a graph $G = (V, E)$ so as to minimize the maximum "stretch" of any edge in the ordering. Formally, let $n = |V|$, and consider any one-to-one mapping $\pi : V \to \{1, 2, \ldots, n\}$. The *bandwidth* of this ordering is $\mathsf{bw}_\pi(G) = \max_{\{u,v\}\in E} |\pi(u) - \pi(v)|$. The *bandwidth of* G is given by the bandwidth of the best possible ordering: $\mathsf{bw}(G) = \min_\pi \mathsf{bw}_\pi(G)$.

The original motivation for this problem lies in the preprocessing of sparse symmetric square matrices. Let A be such an $n \times n$ matrix, and consider the problem of finding a permutation matrix P such that the non-zero entries of $P^{\mathrm{T}}AP$ all lie in as narrow a band as possible about the diagonal. This problem is equivalent to minimizing the bandwidth of the graph G whose vertex set is $\{1, 2, \ldots, n\}$ and which has an edge $\{u, v\}$ precisely when $A_{u,v} \neq 0$.

In lieu of this fact, one tries to efficiently compute a linear ordering π for which $\mathsf{bw}_\pi(G) \le A \cdot \mathsf{bw}(G)$, with the *approximation factor* A is as small as possible. There is even evidence that achieving any value $A = O(1)$ is NP-hard [18]. Much of the difficulty of the bandwidth problem is due to the objective function being a maximum over all edges of the graph. This makes divide-and-conquer approaches ineffective for graph bandwidth, whereas they often succeed for related problems like Minimum Linear Arrangement [6] (here the objective is to minimize $\sum_{\{u,v\}\in E} |\pi(u) - \pi(v)|$). Instead, a more global algorithm is required. To this end, a good lower bound on the value of $\mathsf{bw}(G)$ has to be initially discussed.

The Local Density

For any pair of vertices $u, v \in V$, let $d(u, v)$ to be the shortest path distance between u and v in the graph G. Then, define $B(v, r) = \{u \in V : d(u, v) \le r\}$ as the *ball of radius* r about a vertex $v \in V$. Finally, the *local density of* G is defined by $D(G) = \max_{v\in V, r\ge 1} |B(v, r)|/(2r)$. It is not difficult to see that $\mathsf{bw}(G) \ge D(G)$. Although it was conjectured that an upper bound of the form $\mathsf{bw}(G) \le \mathrm{poly}(\log n) \cdot D(G)$ holds, it was not proven until the seminal work of Feige [7].

Key Results

Feige proved the following.

Theorem 1 *There is an efficient algorithm that, given a graph $G = (V, E)$ as input, produces a linear ordering $\pi : V \to \{1, 2, \ldots, n\}$ for which $\mathsf{bw}_\pi(G) \le O\left((\log n)^3 \sqrt{\log n \log\log n}\right) \cdot D(G)$. In particular, this provides a $\mathrm{poly}(\log n)$-approximation algorithm for the bandwidth problem in general graphs.*

Feige's algorithmic framework can be described quite simply as follows.

1. Compute a representation $f : V \to \mathbb{R}^n$ of G in Euclidean space.
2. Let $u_1, u_2, \ldots, u_n$ be independent $N(0, 1)$[1] random variables, and for each vertex $v \in V$, compute $h(v) =$

[1] $N(0, 1)$ denotes a standard normal random variable with mean 0 and variance 1.

$\sum_{i=1}^{n} u_i f_i(v)$, where $f_i(v)$ is the ith coordinate of the vector $f(v)$.

3. Sort the vertices by the value $h(v)$, breaking ties arbitrarily, and output the induced linear ordering.

An equivalent characterization of steps (2) and (3) is to choose a uniformly random vector $\mathbf{a} \in S^{n-1}$ from the $(n-1)$-dimensional sphere $S^{n-1} \subseteq \mathbb{R}^n$ and output the linear ordering induced by the values $h(v) = \langle \mathbf{a}, f(v) \rangle$, where $\langle \cdot, \cdot \rangle$ denotes the usual inner product on $\mathbb{R}^n$. In other words, the algorithm first computes a map $f: V \to \mathbb{R}^n$, projects the images of the vertices onto a randomly oriented line, and then outputs the induced ordering; step (2) is the standard way that such a random projection is implemented.

Volume-Respecting Embeddings

The only step left unspecified is (1); the function f has to somehow preserve the structure of the graph G in order for the algorithm to output a low-bandwidth ordering. The inspiration for the existence of such an f comes from the field of *low-distortion metric embeddings* (see, e. g. [2,14]). Feige introduced a generalization of low-distortion embeddings to mappings called *volume respecting embeddings*. Roughly, the map f should be non-expansive, in the sense that $\|f(u) - f(v)\| \le 1$ for every edge $\{u, v\} \in E$, and should satisfy the following property: For any set of k vertices $v_1, \dots, v_k$, the $(k-1)$-dimensional volume of the convex hull of the points $f(v_1), \dots, f(v_k)$ should be as large as possible. The proper value of k is chosen to optimize the performance of the algorithm. Refer to [7,10,11] for precise definitions on volume-respecting embeddings, and a detailed discussion of their construction. Feige showed that a modification of Bourgain's embedding [2] yields a mapping $f: V \to \mathbb{R}^n$ which is good enough to obtain the results of Theorem 1.

The requirement $\|f(u) - f(v)\| \le 1$ for every edge $\{u, v\}$ is natural since $f(u)$ and $f(v)$ need to have similar projections onto the random direction $\mathbf{a}$; intuitively, this suggests that u and v will not be mapped too far apart in the induced linear ordering. But even if $|h(u) - h(v)|$ is small, it may be that many vertices project between $h(u)$ and $h(v)$, causing u and v to incur a large stretch. To prevent this, the images of the vertices should be sufficiently "spread out," which corresponds to the volume requirement on the convex hull of the images.

Applications

As was mentioned previously, the graph bandwidth problem has applications to preprocessing sparse symmetric matrices. Minimizing the bandwidth of matrices helps in improving the efficiency of certain linear algebraic algorithms like Gaussian elimination; see [3,8,17]. Follow-up work has shown that Feige's techniques can be applied to VLSI layout problems [19].

Open Problems

First, state the *bandwidth conjecture* (see, e. g. [13]).

Conjecture: For any n-node graph $G = (V, E)$, one has $\mathrm{bw}(G) = O(\log n) \cdot D(G)$.

The conjecture is interesting and unresolved even in the special case when G is a tree (see [9] for the best results for trees). The best-known bound in the general case follows from [7,10], and is of the form $\mathrm{bw}(G) = O(\log n)^{3.5} \cdot D(G)$. It is known that the conjectured upper bound is best possible, even for trees [4]. One suspects that these combinatorial studies will lead to improved approximation algorithms.

However, the best approximation algorithms, which achieve ratio $O((\log n)^3 (\log \log n)^{1/4})$, are not based on the local density bound. Instead, they are a hybrid of a semi-definite programming approach of [1,5] with the arguments of Feige, and the volume-respecting embeddings constructed in [12,16]. Determining the approximability of graph bandwidth is an outstanding open problem, and likely requires improving both the upper and lower bounds.

Recommended Reading

1. Blum, A., Konjevod, G., Ravi, R., Vempala, S.: Semi-definite relaxations for minimum bandwidth and other vertex-ordering problems. Theor. Comput. Sci. **235**(1), 25–42 (2000), Selected papers in honor of Manuel Blum (Hong Kong, 1998)
2. Bourgain, J.: On Lipschitz embedding of finite metric spaces in Hilbert space. Israel J. Math. **52**(1–2), 46–52 (1985)
3. Chinn, P.Z., Chvátalová, J., Dewdney, A.K., Gibbs, N.E.: The bandwidth problem for graphs and matrices—a survey. J. Graph Theory **6**(3), 223–254 (1982)
4. Chung, F.R.K., Seymour, P.D.: Graphs with small bandwidth and cutwidth. Discret. Math. **75**(1–3), 113–119 (1989). Graph theory and combinatorics, Cambridge (1988)
5. Dunagan, J., Vempala, S.: On Euclidean embeddings and bandwidth minimization. In: Randomization, approximation, and combinatorial optimization, pp. 229–240. Springer (2001)
6. Even, G., Naor, J., Rao, S., Schieber, B.: Divide-and-conquer approximation algorithms via spreading metrics. J. ACM **47**(4), 585–616 (2000)
7. Feige, U.: Approximating the bandwidth via volume respecting embeddings. J. Comput. Syst. Sci. **60**(3), 510–539 (2000)
8. George, A., Liu, J.W.H.: Computer solution of large sparse positive definite systems. Prentice-Hall Series in Computational Mathematics, Prentice-Hall Inc. Englewood Cliffs (1981)

9. Gupta, A.: Improved bandwidth approximation for trees and chordal graphs. J. Algorithms **40**(1), 24–36 (2001)
10. Krauthgamer, R., Lee, J.R., Mendel, M., Naor, A.: Measured descent: A new embedding method for finite metrics. Geom. Funct. Anal. **15**(4), 839–858 (2005)
11. Krauthgamer, R., Linial, N., Magen, A.: Metric embeddings–beyond one-dimensional distortion. Discrete Comput. Geom. **31**(3), 339–356 (2004)
12. Lee, J.R.: Volume distortion for subsets of Euclidean spaces. In: Proceedings of the 22nd Annual Symposium on Computational Geometry, ACM, Sedona, AZ 2006, pp. 207–216.
13. Linial, N.: Finite metric-spaces—combinatorics, geometry and algorithms. In: Proceedings of the International Congress of Mathematicians, vol. III, Beijing, 2002, pp. 573–586. Higher Ed. Press, Beijing (2002)
14. Linial, N., London, E., Rabinovich, Y.: The geometry of graphs and some of its algorithmic applications. Combinatorica **15**(2), 215–245 (1995)
15. Papadimitriou, C.H.: The NP-completeness of the bandwidth minimization problem. Computing **16**(3), 263–270 (1976)
16. Rao, S.: Small distortion and volume preserving embeddings for planar and Euclidean metrics. In: Proceedings of the 15th Annual Symposium on Computational Geometry, pp. 300–306. ACM, New York (1999)
17. Strang, G.: Linear algebra and its applications, 2nd edn. Academic Press [Harcourt Brace Jovanovich Publishers], New York (1980)
18. Unger, W.: The complexity of the approximation of the bandwidth problem. In: 39th Annual Symposium on Foundations of Computer Science, IEEE, 8–11 Oct 1998, pp. 82–91.
19. Vempala, S.: Random projection: A new approach to VLSI layout. In: 39th Annual Symposium on Foundations of Computer Science, IEEE, 8–11 Oct 1998, pp. 389–398.

Graph Coloring

1994; Karger, Motwani, Sudan
1998; Karger, Motwani, Sudan

MICHAEL LANGBERG
Department of Computer Science,
The Open University of Israel,
Raanana, Israel

Keywords and Synonyms

Clique cover

Problem Definition

An independent set in an undirected graph $G = (V, E)$ is a set of vertices that induce a subgraph which does not contain any edges. The size of the maximum independent set in G is denoted by $\alpha(G)$. For an integer k, a k-coloring of G is a function $\sigma : V \to [1 \dots k]$ which assigns colors to the vertices of G. A valid k-coloring of G is a coloring in which each color class is an independent set. The chromatic number $\chi(G)$ of G is the smallest k for which there exists a valid k-coloring of G. Finding $\chi(G)$ is a fundamental NP-hard problem. Hence, when limited to polynomial time algorithms, one turns to the question of estimating the value of $\chi(G)$ or to the closely related problem of *approximate coloring*.

Problem 1 (Approximate coloring)
INPUT: *Undirected graph* $G = (V, E)$.
OUTPUT: *A valid coloring of* G *with* $r \cdot \chi(G)$ *colors, for some approximation ratio* $r \geq 1$.
OBJECTIVE: *Minimize* r.

Let G be a graph of size n. The approximate coloring of G can be solved efficiently within an approximation ratio of $r = O\left(\frac{n(\log\log n)^2}{\log^3 n}\right)$ [12]. This holds also for the approximation of $\alpha(G)$ [8]. These results may seem rather weak, however it is NP-hard to approximate $\alpha(G)$ and $\chi(G)$ within a ratio of $n^{1-\varepsilon}$ for any constant $\varepsilon > 0$ [9,14,21]. Under stronger complexity assumptions, there is some constant $0 < \delta < 1$ such that neither problem can be approximated within a ratio of $n/2^{\log^{\delta} n}$ [17,21]. This chapter will concentrate on the problem of coloring graphs G for which $\chi(G)$ is *small*. As will be seen, in this case the approximation ratio achievable significantly improves.

Vector Coloring of Graphs

The algorithms achieving the best ratios for approximate coloring when $\chi(G)$ is small [1,3,13,15] are all based on the idea of *vector coloring*, introduced by Karger, Motwani, and Sudan [15][1]

Definition 1 A vector k-coloring of a graph is an assignment of unit vectors to its vertices, such that for every edge, the inner product of the vectors assigned to its endpoints is at most (in the sense that it can only be more negative) $-1/(k-1)$.

The *vector chromatic number* $\vec{\chi}(G)$ of G is the smallest k for which there exists a vector k-coloring of G. The vector chromatic number can be formulated as follows:

$$\begin{array}{llll} \vec{\chi}(G) & \text{Minimize} & k & \\ & \text{subject to :} & \langle v_i, v_j\rangle \leq -\dfrac{1}{k-1} & \forall (i,j) \in E \\ & & \langle v_i, v_i\rangle = 1 & \forall i \in V . \end{array}$$

Here, assume that $V = [1, \dots, n]$ and that the vectors $\{v_i\}_{i=1}^{n}$ are in R^n. Every k-colorable graph is also vector

[1]Vector coloring as presented in [15] is closely related to the Lovász θ function [19]. This connection will be discussed shortly.

k-colorable. This can be seen by identifying each color class with one vertex of a perfect $(k-1)$-dimensional simplex centered at the origin. Moreover, unlike the chromatic number, a vector k-coloring (when it exists) can be found in polynomial time using semidefinite programming (up to an arbitrarily small error in the inner products).

Claim 1 (Complexity of vector coloring [15]) *Let $\varepsilon > 0$. If a graph G has a vector k-coloring then a vector $(k+\varepsilon)$-coloring of the graph can be constructed in time polynomial in n and $\log(1/\varepsilon)$.*

One can strengthen Definition 1 to obtain a different notion of vector coloring and the vector chromatic number.

$$\begin{array}{llll} \vec{\chi}_2(G) & \text{Minimize} & k & \\ & \text{subject to:} & \langle v_i, v_j\rangle = -\dfrac{1}{k-1} & \forall (i,j) \in E \\ & & \langle v_i, v_i\rangle = 1 & \forall i \in V \end{array}$$

$$\begin{array}{llll} \vec{\chi}_3(G) & \text{Minimize} & k & \\ & \text{subject to:} & \langle v_i, v_j\rangle = -\dfrac{1}{k-1} & \forall (i,j) \in E \\ & & \langle v_i, v_j\rangle \geq -\dfrac{1}{k-1} & \forall i,j \in V \\ & & \langle v_i, v_i\rangle = 1 & \forall i \in V\,. \end{array}$$

The function $\vec{\chi}_2(G)$ is referred to as the *strict* vector chromatic number of G and is equal to the Lovász θ function on $\bar{G}$ [15,19], where $\bar{G}$ is the *complement* graph of G. The function $\vec{\chi}_3(G)$ is referred to as the *strong* vector chromatic number. An analog to Claim 1 holds for both $\vec{\chi}_2(G)$ and $\vec{\chi}_3(G)$. Let $\omega(G)$ denote the size of the maximum clique in G, it holds that: $\omega(G) \leq \vec{\chi}(G) \leq \vec{\chi}_2(G) \leq \vec{\chi}_3(G) \leq \chi(G)$.

Key Results

In what follows, assume that G has n vertices and maximal degree Δ. The $\tilde{O}(\cdot)$ and $\tilde{\Omega}(\cdot)$ notation are used to suppress polylogarithmic factors. The key result of Karger, Motwani, and Sudan [15] is stated below:

Theorem 1 ([15]) *If $\vec{\chi}(G) = k$ then G can be colored in polynomial time using* $\min\{\tilde{O}(\Delta^{1-2/k}), \tilde{O}(n^{1-3/(k+1)})\}$ *colors.*

As mentioned above, the use of vector coloring in the context of approximate coloring was initiated in [15]. Roughly speaking, once given a vector coloring of G, the heart of the algorithm in [15] finds a large independent set in G. In a nutshell, this independent set corresponds to a set of vectors in the vector coloring which are *close* to one another (and thus by definition cannot share an edge). Combining this with the ideas of Wigderson [20] mentioned below yields Theorem 1.

A description of related work is given below. The first two theorems below appeared prior to the work of Karger, Motwani, and Sudan [15].

Theorem 2 ([20]) *If $\chi(G) = k$ then G can be colored in polynomial time using $O(kn^{1-1/(k-1)})$ colors.*

Theorem 3 ([2]) *If $\chi(G) = 3$ then G can be colored in polynomial time using $\tilde{O}(n^{3/8})$ colors. If $\chi(G) = k \geq 4$ then G can be colored in polynomial time using at most $\tilde{O}(n^{1-1/(k-3/2)})$ colors.*

Combining the techniques of [15] and [2] the following results were obtained for graphs G with $\chi(G) = 3, 4$ (these results were also extended for higher values of $\chi(G)$).

Theorem 4 ([3]) *If $\chi(G) = 3$ then G can be colored in polynomial time using $\tilde{O}(n^{3/14})$ colors.*

Theorem 5 ([13]) *If $\chi(G) = 4$ then G can be colored in polynomial time using $\tilde{O}(n^{7/19})$ colors.*

The currently best-known result for coloring a 3-colorable graph is presented in [1]. In their algorithm, [1] use the strict vector coloring relaxation (*i. e.* $\vec{\chi}_2$) enhanced with certain *odd cycle* constraints.

Theorem 6 ([1]) *If $\chi(G) = 3$ then G can be colored in polynomial time using $O(n^{0.2111})$ colors.*

To put the above theorems in perspective, it is NP-hard to color a 3-colorable graph G with 4 colors [11,16] and a k-colorable graph (for sufficiently large k) with $k^{(\log k)/25}$ colors [17]. Under stronger complexity assumptions (related to the Unique Games Conjecture [18]) for any constant k it is hard to color a k-colorable graph with any constant number of colors [6]. The wide gap between these hardness results and the approximation ratios presented in this section has been a major initiative in the study of approximate coloring.

Finally, the limitations of vector coloring are addressed. Namely, are there graphs for which $\vec{\chi}(G)$ is a poor estimate of $\chi(G)$? One would expect the answer to be "yes" as estimating $\chi(G)$ beyond a factor of $n^{1-\varepsilon}$ is a hard problem. As will be stated below, this is indeed the case (even when $\vec{\chi}(G)$ is small). Some of the results that follow are stated in terms of the maximum independent set $\alpha(G)$ in G. As $\chi(G) \geq n/\alpha(G)$, these results imply a lower bound on $\chi(G)$. Theorem 1 (i) states that the original analysis of [15] is essentially tight. Theorem 1 (ii) presents bounds for the case of $\vec{\chi}(G) = 3$. Theorem 1 (iii) and Theorem 2 present graphs G in which there is an extremely large gap between $\chi(G)$ and the relaxations $\vec{\chi}(G)$ and $\vec{\chi}_2(G)$.

Theorem 7 ([10]) *(i) For every constant $\varepsilon > 0$ and constant $k > 2$, there are infinitely many graphs G with $\vec{\chi}(G) = k$ and $\alpha(G) \leq n/\Delta^{1-2/k-\varepsilon}$ (here $\Delta > n^{\delta}$ for some constant $\delta > 0$). (ii) There are infinitely many graphs G with $\vec{\chi}(G) = 3$ and $\alpha(G) \leq n^{0.843}$. (iii) For some constant c, there are infinitely many graphs G with $\vec{\chi}(G) = O(\log n/\log\log n)$ and $\alpha(G) \leq \log^c n$.*

Theorem 8 ([7]) *For some constant c, there are infinitely many graphs G with $\vec{\chi}_2(G) \leq 2^{\sqrt{\log n}}$ and $\chi(G) \geq n/2^{c\sqrt{\log n}}$.*

Vector colorings, including the Lovász θ function and its variants, have been extensively studied in the context of approximation algorithms for problems other than Problem 1. These include approximating $\alpha(G)$, approximating the Minimum Vertex Cover problem, and combinatorial optimization in the context of random graphs.

Applications

Besides its theoretical significance, graph coloring has several concrete applications that fall under the model of *conflict free* allocation of resources (see for example [4,5]).

Open Problems

By far the major open problem in the context of approximate coloring addresses the wide gap between what is known to be hard and what can be obtained in polynomial time. The case of constant $\chi(G)$ is especially intriguing, as the best-known upper bounds (on the approximation ratio) are polynomial while the lower bounds are of constant nature. Regarding the vector coloring paradigm, a majority of the results stated in Sect. "Key Results" use the weakest form of vector coloring $\vec{\chi}(G)$ in their proof, while stronger relaxations such as $\vec{\chi}_2(G)$ and $\vec{\chi}_3(G)$ may also be considered. It would be very interesting to improve upon the algorithmic results stated above using stronger relaxations, as would a matching analysis of the limitations of these relaxations.

Cross References

► Channel Assignment and Routing in Multi-Radio Wireless Mesh Networks
► Max Cut
► Randomized Rounding
► Sparsest Cut

Recommended Reading

1. Arora, S., Chlamtac, E., Charikar, M.: New approximation guarantee for chromatic number. In: Proceedings of the 38th annual ACM Symposium on Theory of Computing (2006) pp. 215–224.
2. Blum, A.: New approximations for graph coloring. J. ACM **41**(3), 470–516 (1994)
3. Blum, A., Karger, D.: An $\tilde{O}(n^{3/14})$-coloring for 3-colorable graphs. Inf. Process. Lett. **61**(6), 49–53 (1997)
4. Chaitin, G.J.: Register allocation & spilling via graph coloring. In: Proceedings of the 1982 SIGPLAN Symposium on Compiler Construction (1982) pp. 98–105.
5. Chaitin, G.J., Auslander, M.A., Chandra, A.K., Cocke, J., Hopkins, M.E., Markstein, P.W.: Register allocation via coloring. Comp. Lang. **6**, 47–57 (1981)
6. Dinur, I., Mossel, E., Regev, O.: Conditional hardness for approximate coloring. In: Proceedings of the 38th annual ACM Symposium on Theory of Computing (2006) pp. 344–353.
7. Feige, U.: Randomized graph products, chromatic numbers, and the Lovász theta function. Combinatorica **17**(1), 79–90 (1997)
8. Feige, U.: Approximating maximum clique by removing subgraphs. SIAM J. Discret. Math. **18**(2), 219–225 (2004)
9. Feige, U., Kilian, J.: Zero knowledge and the chromatic number. J. Comput. Syst. Sci. **57**, 187–199 (1998)
10. Feige, U., Langberg, M., Schechtman, G.: Graphs with tiny vector chromatic numbers and huge chromatic numbers. SIAM J. Comput. **33**(6), 1338–1368 (2004)
11. Guruswami, V., Khanna, S.: On the hardness of 4-coloring a 3-colorable graph. In: Proceedings of the 15th annual IEEE Conference on Computational Complexity (2000) pp. 188–197.
12. Halldorsson, M.: A still better performance guarantee for approximate graph coloring. Inf. Process. Lett. **45**, 19–23 (1993)
13. Halperin, E., Nathaniel, R., Zwick, U.: Coloring k-colorable graphs using smaller palettes. J. Algorithms **45**, 72–90 (2002)
14. Håstad, J.: Clique is hard to approximate within $n^{1-\varepsilon}$. Acta Math. **182**(1), 105–142 (1999)
15. Karger, D., Motwani, R., Sudan, M.: Approximate graph coloring by semidefinite programming. J. ACM **45**(2), 246–265 (1998)
16. Khanna, S., Linial, N., Safra, S.: On the hardness of approximating the chromatic number. Combinatorica **20**, 393–415 (2000)
17. Khot, S.: Improved inapproximability results for max clique, chromatic number and approximate graph coloring. In: Proceedings of the 42nd annual IEEE Symposium on Foundations of Computer Science (2001) pp. 600–609.
18. Khot, S.: On the power of unique 2-prover 1-round games. In: Proceedings of the 34th annual ACM symposium on Theory of Computing (2002) pp. 767–775.
19. Lovász, L.: On the Shannon capacity of a graph. IEEE Trans. Inf. Theor. **25**, 2–13 (1979)
20. Wigderson, A.: Improving the performance guarantee for approximate graph coloring. J. ACM **30**(4), 729–735 (1983)

21. Zuckerman, D.: Linear degree extractors and the inapproximability of max clique and chromatic number. In: Proceedings of the 38th annual ACM symposium on Theory of Computing (2006) pp. 681–690.

Graph Connectivity

1994; Khuller, Vishkin

Samir Khuller[1], Balaji Raghavachari[2]
[1] Computer Science Department, University of Maryland, College Park, MD, USA
[2] Computer Science Department, University of Texas at Dallas, Richardson, TX, USA

Keywords and Synonyms

Highly connected subgraphs; Sparse certificates

Problem Definition

An undirected graph is said to be k-connected (specifically, k-vertex-connected) if the removal of any set of $k-1$ or fewer vertices (with their incident edges) does not disconnect G. Analogously, it is k-edge-connected if the removal of any set of $k-1$ edges does not disconnect G. Menger's theorem states that a k-vertex-connected graph has at least k openly vertex-disjoint paths connecting every pair of vertices. For k-edge-connected graphs there are k edge-disjoint paths connecting every pair of vertices. The connectivity of a graph is the largest value of k for which it is k-connected. Finding the connectivity of a graph, and finding k disjoint paths between a given pair of vertices can be found using algorithms for maximum flow. An edge is said to be *critical* in a k-connected graph if upon its removal the graph is no longer k-connected.

The problem of finding a minimum-cardinality k-vertex-connected (k-edge-connected) subgraph that spans all vertices of a given graph is called k-VCSS (k-ECSS) and is known to be nondeterministic polynomial-time hard for $k \geq 2$. We review some results in finding approximately minimum solutions to k-VCSS and k-ECSS. We focus primarily on simple graphs. A simple approximation algorithm is one that considers the edges in some order and removes edges that are not critical. It thus outputs a k-connected subgraph in which all edges are critical and it can be shown that it is a 2-approximation algorithm (that outputs a solution with at most kn edges in an n-vertex graph, and since each vertex has to have degree at least k, we can claim that $kn/2$ edges are necessary).

Approximation algorithms that do better than the simple algorithm mentioned above can be classified into two categories: depth first search (DFS) based, and matching based.

Key Results

Lower Bounds for k-Connected Spanning Subgraphs

Each node of a k-connected graph has at least k edges incident to it. Therefore, the sum of the degrees of all its nodes is at least kn, where n is the number of its nodes. Since each edge is counted twice in this degree-sum, the cardinality of its edges is at least $kn/2$. This is called the *degree lower bound*. Expanding on this idea yields a stronger lower bound on the cardinality of a k-connected spanning subgraph of a given graph. Let D_k be a subgraph in which the degree of each node is at least k. Unlike a k-connected subgraph, D_k has no connectivity constraints. The counting argument above shows that any D_k has at least $kn/2$ edges. A minimum cardinality D_k can be computed in polynomial time by reducing the problem to matching, and it is called the *matching lower bound*.

DFS-Based Approaches

The following natural algorithm finds a 3/2 approximation for 2-ECSS. Root the tree at some node r and run DFS. All edges of the graph are now either tree edges or back edges. Process the DFS tree in postorder. For each subtree, if the removal of the edge from its root to its parent separates the graph into two components, then add a farthest-back edge from this subtree, whose other end is closest to r. It can be shown that the number of back edges added by the algorithm is at most half the size of *Opt*.

This algorithm has been generalized to solve the 2-VCSS problem with the same approximation ratio, by adding carefully chosen back edges that allow the deletion of tree edges. Wherever it is unable to delete a tree edge, it adds a vertex to an independent set I. In the final analysis, the number of edges used is less than $n + |I|$. Since *Opt* is at least $\max(n, 2|I|)$, it obtains a 3/2-approximation ratio.

The algorithm can also be extended to the k-ECSS problem by repeating these ideas $k/2$ times, augmenting the connectivity by 2 in each round. It has been shown that this algorithm achieves a performance of about 1.61.

Matching-Based Approaches

Several approximation algorithms for k-ECSS and k-VCSS problems have used a minimum cardinality D_k as a starting solution, which is then augmented with additional edges to satisfy the connectivity constraints. This approach yields better ratios than the DFS-based approaches.

$1+\frac{1}{k}$ Algorithm for k-VCSS Find a minimum cardinality D_{k-1}. Add just enough additional edges to it to make the subgraph k-connected. In this step, it is ensured that the edges added are critical. It is known by a theorem of Mader that in a k-connected graph, a cycle of critical edges contains at least one node of degree k. Since the edges added by the algorithm in the second step are all critical, there can be no cycle induced by these edges because the degree of all the nodes on such a cycle would be at least $k+1$. Therefore, at most $n-1$ edges are added in this step. The number of edges added in the first step, in the minimum D_{k-1} is at most $Opt - n/2$. The total number of edges in the solution thus computed is at most $(1 + 1/k)$ times the number of edges in an optimal k-VCSS.

$1+\frac{2}{k+1}$ Algorithm for k-ECSS Mader's theorem about cycles induced by critical edges is valid only for vertex connectivity and not edge connectivity, Therefore, a different algorithm is proposed for k-ECSS in graphs that are k-edge-connected, but not k-connected. This algorithm finds a minimum cardinality D_k and augments it with a minimal set of edges to make the subgraph k-edge-connected. The number of edges added in the last step is at most $\frac{k}{k+1}(n-1)$. Since the number of edges added in the first step is at most Opt, the total number of edges is at most $(1+\frac{2}{k+1})Opt$.

Better Algorithms for Small k For $k \in \{2,3\}$, better algorithms have been obtained by implementing the abovementioned algorithms carefully, deleting unnecessary edges, and by getting better lower bounds. For $k = 2$, a 4/3 approximation can be obtained by generating a path/cycle cover from a minimum cardinality D_2 and 2-connecting them one at a time to a "core" component. Small cycles/paths allow an edge to be deleted when they are 2-connected to the core, which allows a simple amortized analysis. This method also generalizes to the 3-ECSS problem, yielding a 4/3 ratio.

Hybrid approaches have been proposed which use the path/cycle cover to generate a specific DFS tree of the original graph and then 2-connect the tree, trying to delete edges wherever possible. The best ratios achieved using this approach are 5/4 for 2-ECSS, 9/7 for 2-VCSS, and 5/4 for 2-VCSS in 3-connected graphs.

Applications

Network design is one of the main application areas for this work. This involves the construction of low-cost highly connected networks.

Recommended Reading

For additional information on DFS, matchings and path/cycle covers, see [3]. Fast 2-approximation algorithms for k-ECSS and k-VCSS were studied by Nagamochi and Ibaraki [13]. DFS-based algorithms for 2-connectivity were introduced by Khuller and Vishkin [11]. They obtained 3/2 for 2-ECSS, 5/3 for 2-VCSS, and 2 for weighted k-ECSS. The ratio for 2-VCSS was improved to 3/2 by Garg et al. [6], 4/3 by Vempala and Vetta [14], and 9/7 by Gubbala and Raghavachari [7]. Khuller and Raghavachari [10] gave an algorithm for k-ECSS, which was later improved by Gabow [4], who showed that the algorithm obtains a ratio of about 1.61. Cheriyan et al. [2] studied the k-VCSS problem with edge weights and designed an $O(\log k)$ approximation algorithm in graphs with at least $6k^2$ vertices.

The matching-based algorithms were introduced by Cheriyan and Thurimella [1]. They proposed algorithms with ratios of $1+\frac{1}{k}$ for k-VCSS, $1+\frac{2}{k+1}$ for k-ECSS, $1+\frac{1}{k}$ for k-VCSS in directed graphs, and $1+\frac{4}{\sqrt{k}}$ for k-ECSS in directed graphs. Vempala and Vetta [14] obtained a ratio of 4/3 for 2-VCSS. The ratios were further improved by Krysta and Kumar [12], who introduced the hybrid approach, which was used to derive a 5/4 algorithm by Jothi et al. [9]. A 3/2-approximation algorithm for 3-ECSS has been proposed by Gabow [5] that works on multigraphs, whereas the earlier algorithm of Cheriyan and Thurimella gets the same ratio in simple graphs only. This ratio has been improved to 4/3 by Gubbala and Raghavachari [8].

1. Cheriyan, J., Thurimella, R.: Approximating minimum-size k-connected spanning subgraphs via matching. SIAM J. Comput. **30**(2), 528–560 (2000)
2. Cheriyan, J., Vempala, S., Vetta, A.: An approximation algorithm for the minimum-cost k-vertex connected subgraph. SIAM J. Comput. **32**(4), 1050–1055 (2003)
3. Cook, W.J., Cunningham, W.H., Pulleyblank, W.R., Schrijver, A.: Combinatorial optimization. Wiley, New York (1998)
4. Gabow, H.N.: Better performance bounds for finding the smallest k-edge connected spanning subgraph of a multigraph. In: SODA, 2003, pp. 460–469
5. Gabow, H.N.: An ear decomposition approach to approximating the smallest 3-edge connected spanning subgraph of a multigraph. SIAM J. Discret. Math. **18**(1), 41–70 (2004)
6. Garg, N., Vempala, S., Singla, A.: Improved approximation algorithms for biconnected subgraphs via better lower bounding techniques. In: SODA, 1993, pp. 103–111
7. Gubbala, P., Raghavachari, B.: Approximation algorithms for the minimum cardinality two-connected spanning subgraph problem. In: Jünger, M., Kaibel, V. (eds.) IPCO. Lecture Notes in Computer Science, vol. 3509, pp. 422–436. Springer, Berlin (2005)
8. Gubbala, P., Raghavachari, B.: A 4/3-approximation algorithm for minimum 3-edge-connectivity. In: Proceedings of the

Workshop on Algoriths and Data Structures (WADS) August 2007, pp. 39–51. Halifax (2007)
9. Jothi, R., Raghavachari, B., Varadarajan, S.: A 5/4-approximation algorithm for minimum 2-edge-connectivity. In: SODA, 2003, pp. 725–734
10. Khuller, S., Raghavachari, B.: Improved approximation algorithms for uniform connectivity problems. J. Algorithms **21**(2), 434–450 (1996)
11. Khuller, S., Vishkin, U.: Biconnectivity approximations and graph carvings. J. ACM **41**(2), 214–235 (1994)
12. Krysta, P., Kumar, V.S.A.: Approximation algorithms for minimum size 2-connectivity problems. In: Ferreira, A., Reichel, H. (eds.) STACS. Lecture Notes in Computer Science, vol. 2010, pp. 431–442. Springer, Berlin (2001)
13. Nagamochi, H., Ibaraki, T.: A linear-time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph. Algorithmica **7**(5–6), 583–596 (1992)
14. Vempala, S., Vetta, A.: Factor 4/3 approximations for minimum 2-connected subgraphs. In: Jansen, K., Khuller, S. (eds.) APPROX. Lecture Notes in Computer Science, vol. 1913, pp. 262–273. Springer, Berlin (2000)

Graph Isomorphism

1980; McKay

BRENDAN D. MCKAY
Department of Computer Science, Australian National University, Canberra, ACT, Australia

Keywords and Synonyms

Graph matching; Symmetry group

Problem Definition

The problem of determining isomorphism of two combinatorial structures is a ubiquitous one, with applications in many areas. The paradigm case of concern in this chapter is isomorphism of two graphs. In this case, an isomorphism consists of a bijection between the vertex sets of the graphs which induces a bijection between the edge sets of the graphs. One can also take the second graph to be a copy of the first, so that isomorphisms map a graph onto themselves. Such isomorphisms are called *automorphisms* or, less formally, *symmetries*. The set of all automorphisms forms a group under function composition called the *automorphism group*. Computing the automorphism group is a problem rather similar to that of determining isomorphisms.

Graph isomorphism is closely related to many other types of isomorphism of combinatorial structures. In the section entitled "Applications", several examples are given.

Formal Description

A *graph* is a pair $G = (V, E)$ of finite sets, with E being a set of 2-tuples (v, w) of elements of V. The elements of V are called *vertices* (also *points*, *nodes*), while the elements of E are called *directed edges* (also *arcs*). A complementary pair $(v, w), (w, v)$ of directed edges $(v \neq w)$ will be called an *undirected edge* and denoted $\{v, w\}$. A directed edge of the form (v, v) will also be considered an undirected edge, called a *loop* (also *self-loop*). The word "edges" without qualification will indicate undirected edges, directed edges, or both.

Given two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, an *isomorphism* from G_1 to G_2 is a bijection from V_1 to V_2 such that the induced action on E_1 is a bijection onto E_2. If $G_1 = G_2$, then the isomorphism is an *automorphism* of G_1. The set of all *automorphisms* of G_1 is a group under function composition, called the *automorphism group* of G_1, and denoted $\mathrm{Aut}(G_1)$.

In Fig. 1 two isomorphic graphs are shown, together with an isomorphism between them and the automorphism group of the first.

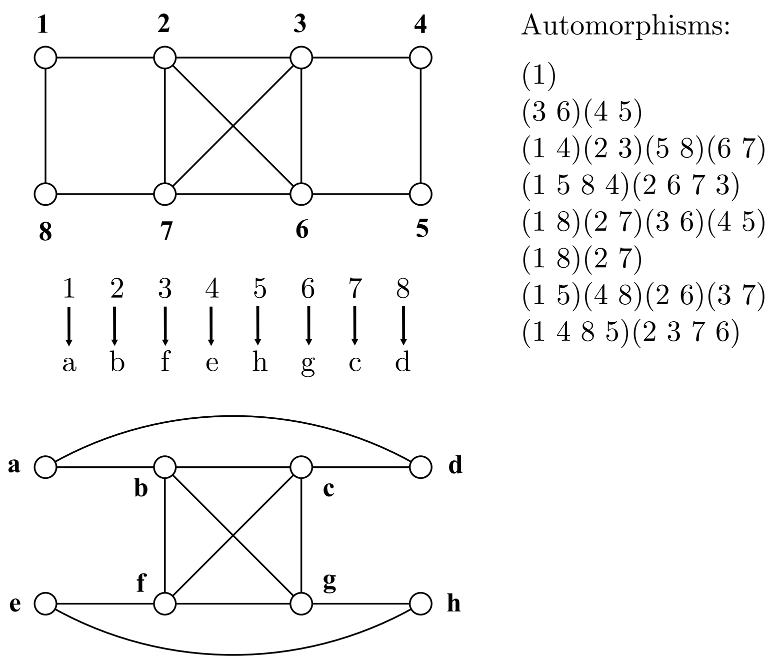
Canonical Labeling

Practical applications of graph isomorphism testing do not usually involve individual pairs of graphs. More commonly, one must decide whether a certain graph is isomorphic to any of a collection of graphs (the database lookup problem) or one has a collection of graphs and needs to identify the isomorphism classes in it (the graph sorting problem). Such applications are not well served by an algorithm that can only test graphs in pairs.

An alternative is a *canonical labeling* algorithm. The essential idea is that in each isomorphism class there is a unique, *canonical* graph which the algorithm can find, given as input any graph in the isomorphism class. The canonical graph might be, for example, the least graph in the isomorphism class according to some ordering (such as lexicographic) of the graphs in the class. Practical algorithms usually compute a canonical form designed for efficiency rather than ease of description.

Key Results

The graph isomorphism problem plays a key role in modern complexity theory. It is not known to be solvable in polynomial time, nor to be NP-complete, nor is it known to be in the class co-NP. See [3,8] for details. Polynomial-time algorithms are known for many special classes, notably graphs with bounded genus, bounded degree, bounded tree-width, and bounded eigenvalue multi-



Graph Isomorphism, Figure 1
Example of an isomorphism and an automorphism group

plicity. The fastest theoretical algorithm for general graphs requires $\exp(n^{1/2+o(1)})$ time [1], but it is not known to be practical.

In this entry, the focus is on the program nauty, which is generally regarded as the most successful for practical use. McKay wrote the first version of nauty in 1976 and described its method of operation in [5]. It is known [7] to have exponential worst-case time, but in practice the worst case is rarely encountered.

The input to nauty is a graph with colored vertices. Two outputs are produced. The first is a set of generators for the color-preserving automorphism group. Though it is rarely necessary, the full group can also be developed element by element. The second, optional, output is a canonical graph. The canonical graph has the following property: two input graphs with the same number of vertices of each color have the same canonical graph if and only if they are isomorphic by a color-preserving isomorphism.

Two graph data structures are supported: a packed adjacency matrix suitable for small dense graphs and a linked list suitable for large sparse graphs.

Applications

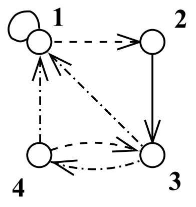
As mentioned, nauty can handle graphs with colored vertices. In this section, it is described how several other types of isomorphism problems can be solved by mapping them onto a problem for vertex-colored graphs.

Isomorphism of Edge-Colored Graphs

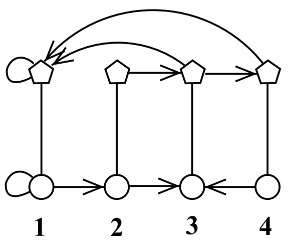
An isomorphism of two graphs, each with both vertices and edges colored, is defined in the obvious way. An example of such a graph appears at the left of Fig. 2.

In the center of the figure the colors are identified with the integers $1, 2, 3$. At the right of the figure an equivalent vertex-colored graph is shown. In this case there are two layers, each with its own color. Edges of color 1 are represented as an edge in the first (lowest) layer, edges of color 2 are represented as an edge in the second layer, and edges of color 3 are represented as edges in both layers. It is now easy to see that the automorphism group of the new graph (specifically, its action on the first layer) is the automorphism group of the original graph. Moreover, the order in which a canonical labeling of the new graph labels the vertices of the first layer can be taken to be a canonical labeling of the original graph.

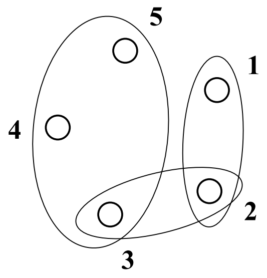
More generally, if the edge colors are integers in $\{1, 2, \ldots, 2^d - 1\}$, there are d layers, and the binary expansion of each color number dictates which layers contain edges. The vertical threads (each corresponding to one vertex of the original graph) can be connected using either paths or cliques. If the original graph has n vertices and k colors, the new graph has $O(n \log k)$ vertices. This can be improved to $O(n\sqrt{\log k})$ vertices by also using edges that are not horizontal.



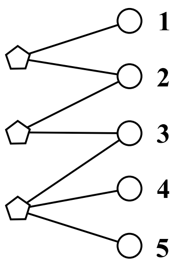
$$\begin{pmatrix} 3 & 1 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 2 & 0 & 0 & 2 \\ 2 & 0 & 1 & 0 \end{pmatrix}$$



Graph Isomorphism, Figure 2
Graph isomorphism with colored edges



$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix}$$



Graph Isomorphism, Figure 3
Hypergraph/design isomorphism as graph isomorphism

Isomorphism of Hypergraphs and Designs

A *hypergraph* is similar to an undirected graph except that the edges can be vertex sets of any size, not just of size 2. Such a structure is also called a *design*.

On the left of Fig. 3 there is a hypergraph with five vertices, two edges of size 2, and one edge of size 3. On the right is an equivalent vertex-colored graph. The vertices on the left, colored with one color, represent the hypergraph edges, while the edges on the right, colored with a different color, represent the hypergraph vertices. The edges of the graph indicate the hypergraph incidence (containment) relationship.

The edge-vertex incidence matrix appears in the center of the figure. This can be any binary matrix at all, which correctly suggests that the problem under consideration is just that of determining the 0-1 matrix equivalence under independent permutation of the rows and columns. By combining this idea with the previous construction, such an equivalence relation on the set of matrices with arbitrary entries can be handled.

Other Examples

For several applications to equivalence operations such as isotopy, important for Latin squares and quasigroups, see [6].

Another important type of equivalence relates matrices over $\{-1, +1\}$. As well as permuting rows and columns, it allows multiplication of rows and columns by -1. A method of converting this *Hadamard equivalence* problem to a graph isomorphism problem is given in [4].

Experimental Results

`Nauty` gives a choice of sparse and dense data structures, and some special code for difficult graph classes. For the following timing examples, the best of the various options are used for a single CPU of a 2.4 GHz Intel Core-duo processor.

1. Random graph with 10,000 vertices, $p = \frac{1}{2}$: 0.014 s for group only, 0.4 s for canonical labeling as well.
2. Random cubic graph with 100,000 vertices: 8 s.
3. 1-skeleton of 20-dimensional cube (1,048,576 vertices, group size 2.5×10^{24}): 92 s.
4. 3-dimensional mesh of size 50 (125,000 vertices): 0.7 s.
5. 1027-vertex strongly regular graph from random Steiner triple system: 0.6 s.

Examples of more difficult graphs can be found in the `nauty` documentation.

URL to Code

The source code of `nauty` is available at http://cs.anu.edu.au/~bdm/nauty/. Another implementation of the automorphism group portion of nauty, highly optimized for large sparse graphs, is available as `saucy` [2]. `Nauty` is also incorporated into a number of general-purpose packages, including GAP, Magma, and MuPad.

Cross References

▶ Abelian Hidden Subgroup Problem
▶ Parameterized Algorithms for Drawing Graphs

Recommended Reading

1. Babai, L., Luks, E.: Canonical labelling of graphs. In: Proceedings of the 15th Annual ACM Symposium on Theory of Computing, pp. 171–183. ACM, New York (1983)
2. Darga, P.T., Liffiton, M.H., Sakallah, K.A., Markov, I.L.: Exploiting Structure in Symmetry Generation for CNF. In: Proceedings of the 41st Design Automation Conference, 2004, pp. 530–534. Source code at http://vlsicad.eecs.umich.edu/BK/SAUCY/
3. Köbler, J., Schöning, U., Torán, J.: The Graph Isomorphism Problem: its structural complexity. Birkhäuser, Boston (1993)
4. McKay, B.D.: Hadamard equivalence via graph isomorphism. Discret. Math. **27**, 213–214 (1979)
5. McKay, B.D.: Practical graph isomorphism. Congr. Numer. **30**, 45–87 (1981)
6. McKay, B.D., Meynert, A., Myrvold, W.: Small Latin squares, quasigroups and loops. J. Comb. Des. **15**, 98–119 (2007)
7. Miyazaki, T.: The complexity of McKay's canonical labelling algorithm. In: Groups and Computation, II. DIMACS Ser. Discret. Math. Theor. Comput. Sci., vol. 28, pp. 239–256. American Mathematical Society, Providence, RI (1997)
8. Toran, J.: On the hardness of graph isomorphism. SIAM J. Comput. **33**, 1093–1108 (2004)

Graphs

▶ Algorithms for Spanners in Weighted Graphs
▶ Minimum Bisection
▶ Mobile Agents and Exploration

Greedy Approximation Algorithms

2004; Ruan, Du, Jia, Wu, Li, Ko

FENG WANG[1], WEILI WU[2]
[1] Department of Mathmatical Science and Applied Computing, Arizona State University, Phoenix, AZ, USA
[2] Department of Computer Science, University of Texas at Dallas, Richardson, TX, USA

Keywords and Synonyms

Technique for analysis of greedy approximation

Problem Definition

Consider a graph $G = (V, E)$. A subset C of V is called a *dominating set* if every vertex is either in C or adjacent to a vertex in C. If, furthermore, the subgraph induced by C is connected, then C is called a *connected dominating set*.

Given a connected graph G, find a connecting dominating set of minimum cardinality. This problem is denoted by MCDS and is NP-hard. Its optimal solution is called a *minimum connected dominating set*. The following is a greedy approximation with potential function f.

Greedy Algorithm A:
```
$C \leftarrow \emptyset$;
**while** $f(C) > 2$ **do**
    choose a vertex $x$ to maximize $f(C) - f(C \cup \{x\})$ and
    $C \leftarrow C \cup \{x\}$; output $C$.
```

Here, f is defined as $f(C) = p(C) + q(C)$ where $p(C)$ is the number of connected components of subgraph induced by C and $q(C)$ is the number of connected components of subgraph with vertex set V and edge set $\{(u, v) \in E \mid u \in C \text{ or } v \in C\}$. f has an important property that C is a connected dominating set if and only if $f(C) = 2$.

If C is a connected dominating set, then $p(C) = q(C) = 1$ and hence $f(C) = 2$. Conversely, suppose $f(C \cup \{x\}) = 2$. Since $p(C) \geq 1$ and $q(C) \geq 1$, one has $p(C) = q(C) = 1$ which implies that C is a connected dominating set. f has another property, for G with at least three vertices, that if $f(C) > 2$, then there exists $x \in V$ such that $f(C) - f(C \cup \{x\}) > 0$. In fact, for $C = \emptyset$, since G is a connected graph with at least three vertices, there must exist a vertex x with degree at least two and for such a vertex x, $f(C \cup \{x\}) < f(C)$. For $C \neq \emptyset$, consider a connected component of the subgraph induced by C. Let B denote its vertex set which is a subset of C. For every vertex y adjacent to B, if y is adjacent to a vertex not adjacent to B and not in C, then $p(C \cup \{y\}) < p(C)$ and $q(C \cup \{y\}) \leq q(C)$; if y is adjacent to a vertex in $C - B$, then $p(C \cup \{y\}) \leq p(C)$ and $q(C \cup \{y\}) < q(C)$.

Now, look at a possible analysis for the above greedy algorithm: Let $x_1, \ldots, X_g$ be vertices chosen by the greedy algorithm in the ordering of their appearance in the algorithm. Denote $C_i = \{x_1, \ldots, x_i\}$. Let $C^* = \{y_1, \ldots, y_{opt}\}$ be a minimum connected dominating set. Since adding C^* to C_i will reduce the potential function value from $f(C_i)$ to 2, the value of f reduced by a vertex in C^* would be $(f(C_i) - 2)/opt$ in average. By the greedy rule for choosing $x_i + 1$, one has

$$f(C_i) - f(C_{i+1}) \geq \frac{f(C_i) - 2}{opt} .$$

Hence,

$$f(C_{i+1}) - 2 \le (f(C_i) - 2)(1 - \frac{1}{opt})$$
$$\le (f(\emptyset) - 2)(1 - \frac{1}{opt})^{i+1} = (n-2)(1 - \frac{1}{opt})^{i+1} ,$$

where $n = |V|$. Note that $1 - 1/opt \le e^{-1/opt}$. Hence,

$$f(C_i) - 2 \le (n-2)\,e^{-i/opt} .$$

Choose i such that $f(C_i) \ge opt + 2 > f(C_{i+1})$. Then

$$opt \le (n-2)\,e^{-i/opt}$$

and

$$g - i \le opt .$$

Therefore,

$$g \le opt + i \le opt\left(1 + \ln\frac{n-2}{opt}\right) .$$

Is this analysis correct? The answer is NO. Why? How could one give a correct analysis. This article will answer those questions and introduce a new general technique, analysis of greedy approximation with nonsubmodular potential function.

Key Results

The Role of Submodularity

Consider a set X and a function f defined on the power set 2^X, i. e., the family of all subsets of X. f is said to be *submodular* if for any two subsets A and B in 2^X,

$$f(A) + f(B) \ge f(A \cap B) + f(A \cup B) .$$

For example, consider a connected graph G. Let X be the vertex set of G. The function $-q(C)$ defined in last section is submodular. To see this, first mention a property of submodular functions.

A submodular function f is *normalized* if $f(\emptyset) = 0$. Every submodular function f can be normalized by setting $g(A) = f(A) - f(\emptyset)$. A function f is *monotone increasing* if $f(A) \le f(B)$ for $A \subset B$. Denote $\Delta_x f(A) = f(A \cup \{x\}) - f(A)$.

Lemma 1 *A function $f: 2^X \to R$ is submodular if and only if $\Delta_x f(A) \le \Delta_x f(B)$ for any $x \in X - B$ and $A \subseteq B$. Moreover, f is monotone increasing if and only if $\Delta_x f(A) \le \Delta_x f(B)$ for any $x \in B$ and $A \subseteq B$.*

Proof If f is submodular, then for $x \in X - B$ and $A \subseteq B$, one has

$$f(A \cup \{x\}) + f(B)$$
$$\ge f((A \cup \{x\}) \cup B) + f(A \cup \{x\}) \cap B)$$
$$= f(B \cup \{x\}) + f(A) ,$$

that is,

$$\Delta_x f(A) \ge \Delta_x f(B) . \quad (1)$$

Conversely, suppose (1) holds for any $x \in B$ and $A \subseteq B$. Let C and D be two set and $C \setminus D = \{x_1, \ldots, x_k\}$. Then

$$f(C \cup D) - f(D) = \sum_{i=1}^{k} \Delta_{x_i} f(D \cup \{x_1, \ldots, x_{i-1})$$
$$\le \sum_{i=1}^{k} \Delta_{x_i} f((C \cap D) \cup \{x_1, \ldots, x_{i-1})$$
$$= f(C) - f(C \cap D) .$$

If f is monotone increasing, then for $A \subseteq B$, $f(A) \le f(B)$. Hence, for $x \in B$,

$$\Delta)x f(A) \ge 0 = \Delta_x f(B) .$$

Conversely, if $\Delta_x f(A) \ge \Delta_x f(B)$ for any $x \in B$ and $A \subseteq B$, then for any x and A, $\Delta_x f(A) \ge \Delta_x f(A \cup \{x\}) = 0$, that is $f(A) \le f(A \cup \{x\})$. Let $B - A = \{x_1, \ldots, x_k\}$. Then

$$f(A) \le f(A \cup \{x_1\}) \le f(A \cup \{x_1, x_2\}) \le \cdots \le f(B) .$$

Next, the submodularity of $-q(A)$ is studied.

Lemma 2 *If $A \subset B$, then $\Delta_y q(A) \ge \Delta_y q(B)$.*

Proof Note that each connected component of graph $(V, D(B))$ is constituted by one or more connected components of graph $(V, D(A))$ since $A \subset B$. Thus, the number of connected components of $(V, D(B))$ dominated by y is no more than the number of connected components of $(V, D(A))$ dominated by y. Therefore, the lemma holds.

The relationship between submodular functions and greedy algorithms have been established for a long time [3].

Let f be a normalized, monotone increasing, submodular integer function. Consider the minimization problem

$$\begin{array}{rl} \min & c(A) \\ \text{subject to} & A \in C_f . \end{array}$$

where c is a nonnegative cost function defined on 2^X and $C_f = \{C \mid f(C \cup \{x\}) - f(C) = 0 \text{ for all } x \in X\}$. The following is a greedy algorithm to produce approximation solution for this problem.

Greedy Algorithm B

```
input submodular function $f$ and cost function $c$;
$A \leftarrow \emptyset$;
**while** there exists $x \in E$ such that $\Delta_x f(A) > 0$
**do** select a vertex $x$ that maximizes $\Delta_x f(A)/c(x)$ and set
   $A \leftarrow A \cup \{x\}$;
return $A$.
```

The following two results are well-known.

Theorem 1 *If f is a normalized, monotone increasing, submodular integer function, then Greedy Algorithm B produces an approximation solution within a factor of $H(\gamma)$ from optimal, where $\gamma = \max_{x\in E} f(\{x\})$.*

Theorem 2 *Let f be a normalized, monotone increasing, submodular function and c a nonnegative cost function. If in Greedy Algorithm B, selected x always satisfies $\Delta_x f(A_{i-1})/c(x) \geq 1$, then it produces an approximation solution within a factor of $1 + \ln(f^*/opt)$ from optimal for above minimization problem where $f^* = f(A^*)$ and $opt = c(A^*)$ for optimal solution A^*.*

Now, come back to the analysis of Greedy Algorithm A for the MCDS. It looks like that the submodularity of f is not used. Actually, the submodularity was implicitly used in the following statement:

"Since adding C^* to C_i will reduce the potential function value from $f(C_i)$ to 2, the value of f reduced by a vertex in C^* would be $(f(C_i) - 2)/opt$ in average. By the greedy rule for choosing $x_i + 1$, one has

$$f(C_i) - f(C_{i+1}) \geq \frac{f(C_i) - 2}{opt} \text{ ."}$$

To see this, write this argument more carefully.

Let $C^* = \{y_1, \ldots, y_{opt}\}$ and denote $C_j^* = \{y_1, \ldots, y_j\}$. Then

$$\begin{aligned} f(C_i) - 2 &= f(C_i) - f(C_i \cup C^*) \\ &= \sum_{j=1}^{opt} [f(C_i \cup C_{j-1}^*) - f(C_i \cup C_j^*)] \end{aligned}$$

where $C_0^* = \emptyset$. By the greedy rule for choosing $x_i + 1$, one has

$$f(C_i) - f(C_{i+1}) \geq f(C_i) - f(C_i \cup \{y_j\})$$

for $j = 1, \ldots, opt$. Therefore, it needs to have

$$\begin{aligned} -\Delta_{y_j} f(C_i) &= f(C_i) - f(C_i \cup \{y_j\}) \\ &\geq f(C_i \cup C_{j-1}^*) - f(C_i \cup C_j^*) \\ &= -\Delta_{y_j} f(C_i \cup C_{j-1}^*) \end{aligned} \quad (2)$$

in order to have

$$f(C_i) - f(C_{i+1}) \geq \frac{f(C_i) - 2}{opt} \text{ .}$$

(2) asks the submodularity of $-f$. Unfortunately, $-f$ is not submodular. A counterexample can be found in [3]. This is why the analysis of Greedy Algorithm A in Sect. "Problem Definition" is incorrect.

Giving up Submodularity

Giving up submodularity is a challenge task since it is open for a long time. But, it is possible based on the following observation on (2) by Du et al. [1]: **The submodularity of $-f$ is applied to increment of a vertex y_j belonging to optimal solution** C^*.

Since the ordering of y_j's is flexible, one may arrange it to make $\Delta_{y_j} f(C_i) - \Delta_{y_j} f(C_i \cup C_{j-1}^*)$ under control. This is a successful idea for the MCDS.

Lemma 3 *Let y_j's be ordered in the way that for any $j = 1, \ldots, opt$, $\{y_1, \ldots, y_j\}$ induces a connected subgraph. Then*

$$\Delta_{y_j} f(C_i) - \Delta_{y_j} f(C_i \cup C_{j-1}^*) \leq 1 \text{ .}$$

Proof Since all $y_1, \ldots, y_{j-1}$ are connected, y_j can dominate at most one additional connected component in the subgraph induced by $C_{i-1} \cup C_{j-1}^*$ than in the subgraph induced by $c_i - 1$. Hence

$$\Delta_{y_j} p(C_i) - \Delta_{y_j} f(C_i \cup C_{j-1}^*) \leq 1 \text{ .}$$

Moreover, since $-q$ is submodular,

$$\Delta_{y_j} q(C_i) - \Delta_{y_j} q(C_i \cup C_{j-1}^*) \leq 0 \text{ .}$$

Therefore,

$$\Delta_{y_j} f(C_i) - \Delta_{y_j} f(C_i \cup C_{j-1}^*) \leq 1 \text{ .}$$

Now, one can give a correct analysis for the greedy algorithm for the MCDS [4].

By Lemma 3,

$$f(C_i) - f(C_{i+1}) \geq \frac{f(C_i) - 2}{opt} - 1 \text{ .}$$

Hence,

$$\begin{aligned} f(C_{i+1}) - 2 - opt &\le (f(C_i) - 2 + opt)\left(1 - \frac{1}{opt}\right) \\ &\le (f(\emptyset) - 2 - opt)\left(1 - \frac{1}{opt}\right)^{i+1} \\ &= (n - 2 - opt)\left(1 - \frac{1}{opt}\right)^{i+1}, \end{aligned}$$

where $n = |V|$. Note that $1 - 1/opt \le e^{-1/opt}$. Hence,

$$f(C_i) - 2 - opt \le (n-2)\,e^{-i/opt} .$$

Choose i such that $f(C_i) \ge 2 \cdot opt + 2 > f(C_{i+1})$. Then

$$opt \le (n-2)\,e^{-i/opt}$$

and

$$g - i \le 2 \cdot opt .$$

Therefore,

$$g \le 2 \cdot opt + i \le opt\left(2 + \ln \frac{n-2}{opt}\right) \le opt(2 + \ln \delta)$$

where δ is the maximum degree of input graph G.

Applications

The technique introduced in previous section has many applications, including analysis of iterated 1-Steiner trees for minimum Steiner tree problem and analysis of greedy approximations for optimization problems in optical networks [4] and wireless networks [3].

Open Problems

Can one show the performance ratio $1 + H(\delta)$ for Greedy Algorithm B for the MCDS? The answer is unknown. More generally, it is unknown how to get a clean generalization of Theorem 1.

Cross References

► Connected Dominating Set
► Local Search Algorithms for kSAT
► Steiner Trees

Acknowledgments

Weili Wu is partially supported by NSF grant ACI-0305567.

Recommended Reading

1. Du, D.-Z., Graham, R.L., Pardalos, P.M., Wan, P.-J., Wu, W., Zhao, W.: Analysis of greedy approximations with nonsubmodular potential functions. ACM-SIAM Symposium on Discrete Algorithms (SODA), 2008
3. Nemhauser, G.L., Wolsey, L.A.: Integer and Combinatorial Optimization. Wiley, Hoboken (1999)
4. Ruan, L., Du, H., Jia, X., Wu, W., Li, Y., Ko, K.-I.: A greedy approximation for minimum connected dominating set. Theor. Comput. Sci. **329**, 325–330 (2004)
5. Ruan, L., Wu, W.: Broadcast routing with minimum wavelength conversion in WDM optical networks. J. Comb. Optim. **9** 223–235 (2005)

Greedy Set-Cover Algorithms

1974–1979; Chvátal, Johnson, Lovász, Stein

NEAL E. YOUNG
Department of Computer Science, University of California at Riverside, Riverside, CA, USA

Keywords and Synonyms

Dominating set; Greedy algorithm; Hitting set; Set cover; Minimizing a linear function subject to a submodular constraint

Problem Definition

Given a collection S of sets over a universe U, a *set cover* $C \subseteq S$ is a subcollection of the sets whose union is U. The *set-cover problem* is, given S, to find a minimum-cardinality set cover. In the *weighted set-cover problem*, for each set $s \in S$ a weight $w_s \ge 0$ is also specified, and the goal is to find a set cover C of minimum total weight $\sum_{s \in C} w_s$.

Weighted set cover is a special case of *minimizing a linear function subject to a submodular constraint*, defined as follows. Given a collection S of objects, for each object s a non-negative weight w_s, and a non-decreasing submodular function $f : 2^S \to \mathbb{R}$, the goal is to find a subcollection $C \subseteq S$ such that $f(C) = f(S)$ minimizing $\sum_{s \in C} w_s$. (Taking $f(C) = |\cup_{s \in C} s|$ gives weighted set cover.)

Key Results

The *greedy algorithm* for weighted set cover builds a cover by repeatedly choosing a set s that minimize the weight w_s divided by number of elements in s not yet covered by chosen sets. It stops and returns the chosen sets when they form a cover:

```
greedy-set-cover($S$, $w$)
1. Initialize $C \leftarrow \emptyset$. Define $f(C) \doteq |\cup_{s\in C} s|$.
2. Repeat until $f(C) = f(S)$:
3.     Choose $s \in S$ minimizing the price per
       element $w_s/[f(C \cup \{s\}) - f(C)]$.
4.     Let $C \leftarrow C \cup \{s\}$.
5. Return $C$.
```

Let H_k denote $\sum_{i=1}^{k} 1/i \approx \ln k$, where k is the largest set size.

Theorem 1 *The greedy algorithm returns a set cover of weight at most H_k times the minimum weight of any cover.*

Proof When the greedy algorithm chooses a set s, imagine that it charges the price per element for that iteration to each element newly covered by s. Then the total weight of the sets chosen by the algorithm equals the total amount charged, and each element is charged once.

Consider any set $s = \{x_k, x_{k-1}, \ldots, x_1\}$ in the optimal set cover C^*. Without loss of generality, suppose that the greedy algorithm covers the elements of s in the order given: $x_k, x_{k-1}, \ldots, x_1$. At the start of the iteration in which the algorithm covers element x_i of s, at least i elements of s remain uncovered. Thus, if the greedy algorithm were to choose s in that iteration, it would pay a cost per element of at most w_s/i. Thus, in this iteration, the greedy algorithm pays at most w_s/i per element covered. Thus, it charges element x_i at most w_s/i to be covered. Summing over i, the total amount charged to elements in s is at most $w_s H_k$. Summing over $s \in C^*$ and noting that every element is in some set in C^*, the total amount charged to elements overall is at most $\sum_{s\in C^*} w_s H_k = H_k \mathrm{OPT}$. $\square$

The theorem was shown first for the unweighted case (each $w_s = 1$) by Johnson [6], Lovász [9], and Stein [14], then extended to the weighted case by Chvátal [2].

Since then a few refinements and improvements have been shown, including the following:

Theorem 2 *Let S be a set system over a universe with n elements and weights $w_s \le 1$. The total weight of the cover C returned by the greedy algorithm is at most $[1 + \ln(n/\mathrm{OPT})]\mathrm{OPT} + 1$ (compare to [13]).*

Proof Assume without loss of generality that the algorithm covers the elements in order $x_n, x_{n-1}, \ldots, x_1$. At the start of the iteration in which the algorithm covers x_i, there are at least i elements left to cover, and all of them could be covered using multiple sets of total cost OPT. Thus, there is some set that covers not-yet-covered elements at a cost of at most OPT/i per element.

Recall the charging scheme from the previous proof. By the preceding observation, element x_i is charged at most OPT/i. Thus, the total charge to elements $x_n, \ldots, x_i$ is at most $(H_n - H_{i-1})\mathrm{OPT}$. Using the assumption that each $w_s \le 1$, the charge to each of the remaining elements is at most 1 per element. Thus, the total charge to all elements is at most $i - 1 + (H_n - H_{i-1})\mathrm{OPT}$. Taking $i = 1 + \lceil \mathrm{OPT} \rceil$, the total charge is at most $\lceil \mathrm{OPT} \rceil + (H_n - H_{\lceil \mathrm{OPT} \rceil})\mathrm{OPT} \le 1 + \mathrm{OPT}(1 + \ln(n/\mathrm{OPT}))$. $\square$

Each of the above proofs implicitly constructs a linear-programming primal-dual pair to show the approximation ratio. The same approximation ratios can be shown with respect to any fractional optimum (solution to the fractional set-cover linear program).

Other Results

The greedy algorithm has been shown to have an approximation ratio of $\ln n - \ln\ln n + O(1)$ [12]. For the special case of set systems whose duals have finite Vapnik-Chervonenkis (VC) dimension, other algorithms have substantially better approximation ratio [1]. Constant-factor approximation algorithms are known for geometric variants of the closely related k-median and facility location problems.

The greedy algorithm generalizes naturally to many problems. For example, for minimizing a linear function subject to a submodular constraint (defined above), the natural extension of the greedy algorithm gives an H_k-approximate solution, where $k = \max_{s\in S} f(\{s\}) - f(\emptyset)$, assuming f is integer-valued [10].

The set-cover problem generalizes to allow each element x to require an arbitrary number r_x of sets containing it to be in the cover. This generalization admits a polynomial-time $O(\log n)$-approximation algorithm [8].

The special case when each element belongs to at most r sets has a simple r-approximation algorithm ([15] § 15.2). When the sets have uniform weights ($w_s = 1$), the algorithm reduces to the following: select any maximal collection of elements, no two of which are contained in the same set; return all sets that contain a selected element.

The variant "Max k-coverage" asks for a set collection of total weight at most k covering as many of the elements as possible. This variant has a $(1 - 1/e)$-approximation algorithm ([15] Problem 2.18) (see [7] for sets with non-uniform weights).

For a general discussion of greedy methods for approximate combinatorial optimization, see ([5] Ch. 4).

Finally, under likely complexity-theoretic assumptions, the $\ln n$ approximation ratio is essentially the best possible for any polynomial-time algorithm [3,4].

Applications

Set Cover and its generalizations and variants are fundamental problems with numerous applications. Examples include:

- selecting a small number of nodes in a network to store a file so that all nodes have a nearby copy,
- selecting a small number of sentences to be uttered to tune all features in a speech-recognition model [11],
- selecting a small number of telescope snapshots to be taken to capture light from all galaxies in the night sky,
- finding a short string having each string in a given set as a contiguous sub-string.

Cross References

► Local Search for *K*-medians and Facility Location

Recommended Reading

1. Brönnimann, H., Goodrich, M.T.: Almost optimal set covers in finite VC-dimension. Discret. Comput. Geom. **14**(4), 463–479 (1995)
2. Chvátal, V.: A greedy heuristic for the set-covering problem. Math. Oper. Res. **4**(3), 233–235 (1979)
3. Lund, C., Yannakakis, M.: On the hardness of approximating minimization problems. J. ACM **41**(5), 960–981 (1994)
4. Feige, U.: A threshold of ln n for approximating set cover. J. ACM **45**(4), 634–652 (1998)
5. Gonzalez, T.F.: Handbook of Approximation Algorithms and Metaheuristics. Chapman & Hall/CRC Computer & Information Science Series (2007)
6. Johnson, D.S.: Approximation algorithms for combinatorial problems. J. Comput. Syst. Sci. **9**, 256–278 (1974)
7. Khuller, S., Moss, A., Naor, J.: The budgeted maximum coverage problem. Inform. Process. Lett. **70**(1), 39–45 (1999)
8. Kolliopoulos, S.G., Young, N.E.: Tight approximation results for general covering integer programs. In: Proceedings of the forty-second annual IEEE Symposium on Foundations of Computer Science, pp. 522–528 (2001)
9. Lovász, L.: On the ratio of optimal integral and fractional covers. Discret. Math. **13**, 383–390 (1975)
10. Nemhauser, G.L., Wolsey, L.A.: Integer and Combinatorial Optimization. Wiley, New York (1988)
11. van Santen, J.P.H., Buchsbaum, A.L.: Methods for optimal text selection. In: Proceedings of the European Conference on Speech Communication and Technology (Rhodos, Greece) **2**, 553–556 (1997)
12. Slavik, P.: A tight analysis of the greedy algorithm for set cover. J. Algorithms **25**(2), 237–254 (1997)
13. Srinivasan, A.: Improved approximations of packing and covering problems. In: Proceedings of the twenty-seventh annual ACM Symposium on Theory of Computing, pp. 268–276 (1995)
14. Stein, S.K.: Two combinatorial covering theorems. J. Comb. Theor. A **16**, 391–397 (1974)
15. Vazirani, V.V.: Approximation Algorithms. Springer, Berlin Heidelberg (2001)

H

Hamilton Cycles in Random Intersection Graphs

2005; Efthymiou, Spirakis

CHARILAOS EFTHYMIOU[1], PAUL SPIRAKIS[2]
[1] Department of Computer Engineering and Informatics, University of Patras, Patras, Greece
[2] Computer Engineering and Informatics, Research and Academic Computer Technology Institute, Patras University, Patras, Greece

Keywords and Synonyms

Threshold for appearance of Hamilton cycles in random intersection graphs; Stochastic order relations between Erdös–Rényi random graph model and random intersection graphs

Problem Definition

E. Marczewski proved that every graph can be represented by a list of sets where each vertex corresponds to a set and the edges to nonempty intersections of sets. It is natural to ask what sort of graphs would be most likely to arise if the list of sets is generated randomly.

Consider the model of random graphs where each vertex chooses randomly from a universal set the members of its corresponding set, each independently of the others. The probability space that is created is the space of random intersection graphs, $G_{n,m,p}$, where n is the number of vertices, m is the cardinality of a universal set of elements and p is the probability for each vertex to choose an element of the universal set. The model of random intersection graphs was first introduced by M. Karońsky, E. Scheinerman, and K. Singer-Cohen in [4]. A rigorous definition of the model of random intersection graphs follows:

Definition 1 Let n, m be positive integers and $0 \leq p \leq 1$. The random intersection graph $G_{n,m,p}$ is a probability space over the set of graphs on the vertex set $\{1, \ldots, n\}$ where each vertex is assigned a random subset from a fixed set of m elements. An edge arises between two vertices when their sets have at least a common element. Each random subset assigned to a vertex is determined by

$$\mathbf{Pr}\left[\text{vertex } i \text{ chooses element } j\right] = p$$

with these events mutually independent.

A common question for a graph is whether it has a cycle, a set of edges that form a path so that the first and the last vertex is the same, that visits *all* the vertices of the graph exactly once. We call this kind of cycle the *Hamilton cycle* and the graph that contains such a cycle is called a *Hamiltonian graph*.

Definition 2 Consider an undirected graph $G = (V, E)$ where V is the set of vertices and E the set of edges. This graph contains a Hamilton cycle if and only if there is a simple cycle that contains each vertex in V.

Consider an instance of $G_{n,m,p}$, for specific values of its parameters n, m, and p, what is the probability of that instance to be Hamiltonian? Taking the parameter p, of the model, to be a function of n and m, in [2], a threshold function $P(n, m)$ has been found for the graph property "Contains a Hamilton cycle"; i. e. a function $P(n, m)$ is derived such that

$$\begin{aligned}&\text{if } p(n,m) \ll P(n,m)\\ &\lim_{n,m\to\infty} \mathbf{Pr}\left[G_{n,m,p} \text{ Contains Hamilton cycle}\right] = 0\\ &\text{if } p(n,m) \gg P(n,m)\\ &\lim_{n,m\to\infty} \mathbf{Pr}\left[G_{n,m,p} \text{ Contains Hamilton cycle}\right] = 1\end{aligned}$$

When a graph property, such as "Contains a Hamilton cycle," holds with probability that tends to 1 (or 0) as n, m tend to infinity, then it is said that this property holds (does not hold), "almost surely" or "almost certainly."

If in $G_{n,m,p}$ the parameter m is very small compared to n, the model is not particularly interesting and when m is exceedingly large (compared to n) the behavior of $G_{n,m,p}$ is essentially the same as the Erdös–Rényi model of random graphs (see [3]). If someone takes $m = \lceil n^\alpha \rceil$, for fixed real $\alpha > 0$, then there is some deviation from the standard models, while allowing for a natural progression from sparse to dense graphs. Thus, the parameter m is assumed to be of the form $m = \lceil n^\alpha \rceil$ for some fixed positive real α.

The proof of existence of a Hamilton cycle in $G_{n,m,p}$ is mainly based on the establishment of a *stochastic order relation* between the model $G_{n,m,p}$ and the Erdös–Rényi random graph model $G_{n,\hat{p}}$.

Definition 3 Let n be a positive integer, $0 \le \hat{p} \le 1$. The random graph $G(n, \hat{p})$ is a probability space over the set of graphs on the vertex set $\{1, \ldots, n\}$ determined by

$$\mathbf{Pr}\left[i, j\right] = \hat{p}$$

with these events mutually independent.

The stochastic order relation between the two models of random graphs is established in the sense that if $\mathcal{A}$ is an increasing graph property, then it holds that

$$\mathbf{Pr}\left[G_{n,\hat{p}} \in \mathcal{A}\right] \le \mathbf{Pr}\left[G_{n,m,p} \in \mathcal{A}\right]$$

where $\hat{p} = f(p)$. A graph property $\mathcal{A}$ is increasing if and only if given that $\mathcal{A}$ holds for a graph $G(V, E)$ then $\mathcal{A}$ holds for any $G(V, E')$: $E' \supseteq E$.

Key Results

Theorem 1 *Let $m = \lceil n^\alpha \rceil$, where α is a fixed real positive, and C_1, C_2 be sufficiently large constants. If*

$$p \ge C_1 \frac{\log n}{m} \quad \text{for} \quad 0 < \alpha < 1 \quad \text{or}$$

$$p \ge C_2 \sqrt{\frac{\log n}{nm}} \quad \text{for} \quad \alpha > 1$$

then almost all $G_{n,m,p}$ are Hamiltonian. Our bounds are asymptotically tight.

Note that the theorem above says nothing when $m = n$, i. e. $\alpha = 1$.

Applications

The Erdös–Rényi model of random graphs, $G_{n,p}$, is exhaustively studied in computer science because it provides a framework for studying practical problems such as "reliable network computing" or it provides a "typical instance" of a graph and thus it is used for average case analysis of graph algorithms. However, the simplicity of $G_{n,p}$ means it is not able to capture satisfactorily many practical problems in computer science. Basically, this is because of the fact that in many problems independent edge-events are not well justified. For example, consider a graph whose vertices represent a set of objects that either are placed or move in a specific geographical region, and the edges are radio communication links. In such a graph, we expect that, any two vertices u, w are more likely to be adjacent to each other, than any other, arbitrary, pair of vertices, if both are adjecent to a third vertex v. Even epidemiological phenomena (like the spread of disease) tend to be more accurately captured by this proximity-sensitive random intersection graph model. Other applications may include oblivious resource sharing in a distributive setting, interaction of mobile agents traversing the web etc.

The model of random intersection graphs $G_{n,m,p}$ was first introduced by M. Karoński, E. Scheinerman, and K. Singer-Cohen in [4] where they explored the evolution of random intersection graphs by studying the thresholds for the appearance and disappearance of small induced subgraphs. Also, J.A. Fill, E.R. Scheinerman, and K. Singer Cohen in [3] proved an equivalence theorem relating the evolution of $G_{n,m,p}$ and $G_{n,p}$, in particular they proved that when $m = n^\alpha$ where $\alpha > 6$, the total variation distance between the graph random variables has limit 0. S. Nikoletseas, C. Raptopoulos, and P. Spirakis in [8] studied the existence and the efficient algorithmic construction of close to optimal independent sets in random intersection graphs. D. Stark in [12] studied the degree of the vertices of the random intersection graphs. However, after [2], Spirakis and Raptopoulos, in [11], provide algorithms that construct Hamilton cycles in instances of $G_{n,m,p}$, for p above the Hamiltonicity threshold. Finally, Nikoletseas et.al in [7] study the mixing time and cover time as the parameter p of the model varies.

Open Problems

As in many other random structures, e. g. $G_{n,p}$ and random formulae, properties of random intersection graphs also appear to have threshold behavior. So far threshold behavior has been studied for the induced subgraph appearance and hamiltonicity.

Other fields of research for random intersection graphs may include the study of connectivity behavior, of the model i. e. the path formation, the formation of giant components. Additionally, a very interesting research question is how cover and mixing times vary with the parameter p, of the model.

Cross References

► Independent Sets in Random Intersection Graphs

Recommended Reading

1. Alon, N., Spencer, J.H.: The Probabilistic Method. 2nd edn. Wiley, New York (2000)
2. Efthymiou, C., Spirakis, P.G.: On the Existence of Hamilton Cycles in Random Intersection Graphs. In: Proc. of the 32nd ICALP. LNCS, vol. 3580, pp. 690–701. Springer, Berlin/Heidelberg (2005)
3. Fill, J.A., Scheinerman, E.R., Singer-Cohen, K.B.: Random intersection graphs when $m = \omega(n)$: an equivalence theorem relating the evolution of the $G(n, m, p)$ and $G(n, p)$ models. Random Struct. Algorithms **16**, 156–176 (2000)
4. Karoński, M., Scheinerman, E.R., Singer-Cohen, K.: On Random Intersection Graphs: The Subgraph Problem. Comb. Probab. Comput. **8**, 131–159 (1999)
5. Komlós, J., Szemerédi, E.: Limit Distributions for the existence of Hamilton cycles in a random graph. Discret. Math. **43**, 55–63 (1983)
6. Korshunov, A.D.: Solution of a problem of P. Erdös and A. Rényi on Hamilton Cycles in non-oriented graphs. Metody Diskr. Anal. Teoriy Upr. Syst. Sb. Trubov Novosibrirsk **31**, 17–56 (1977)
7. Nikoletseas, S., Raptopoulos, C., Spirakis, P.: Expander Properties and the Cover Time of Random Intersection Graphs. In: Proc of the 32nd MFCS, pp. 44–55. Springer, Berlin/Heidelberg (2007)
8. Nikoletseas, S., Raptopoulos, C., Spirakis, P.: The existence and Efficient construction of Large Independent Sets in General Random Intersection Graphs. In: Proc. of the 31st ICALP. LNCS, vol. 3142, pp. 1029–1040. Springer, Berlin/Heidelberg (2004)
10. Singer, K.: Random Intersection Graphs. Ph. D. thesis, The Johns Hopkins University, Baltimore (1995)
11. Spirakis, P.G. Raptopoulos, C.: Simple and Efficient Greedy Algorithms for Hamilton Cycles in Random Intersection Graphs. In: Proc. of the 16th ISAAC. LNCS, vol. 3827, pp. 493–504. Springer, Berlin/Heidelberg (2005)
12. Stark, D.: The Vertex Degree Distribution of Random Intersection Graphs. Random Struct. Algorithms **24**, 249–258 (2004)

Hardness of Proper Learning

1988; Pitt, Valiant

VITALY FELDMAN
Department of Engineering and Applied Sciences, Harvard University, Cambridge, MA, USA

Keywords and Synonyms

Representation-based hardness of learning

Problem Definition

The work of Pitt and Valiant [16] deals with learning Boolean functions in the Probably Approximately Correct (PAC) learning model introduced by Valiant [17]. A learning algorithm in Valiant's original model is given random examples of a function $f : \{0,1\}^n \to \{0,1\}$ from a representation class $\mathcal{F}$ and produces a hypothesis $h \in \mathcal{F}$ that closely approximates f. Here a *representation class* is a set of functions and a language for describing the functions in the set. The authors give examples of natural representation classes that are NP-hard to learn in this model whereas they can be learned if the learning algorithm is allowed to produce hypotheses from a richer representation class $\mathcal{H}$. Such an algorithm is said to learn $\mathcal{F}$ by $\mathcal{H}$; learning $\mathcal{F}$ by $\mathcal{F}$ is called *proper* learning.

The results of Pitt and Valiant were the first to demonstrate that the choice of representation of hypotheses can have a dramatic impact on the computational complexity of a learning problem. Their specific reductions from NP-hard problems are the basis of several other follow-up works on the hardness of proper learning [1,3,6].

Notation

Learning in the PAC model is based on the assumption that the unknown function (or *concept*) belongs to a certain class of concepts C. In order to discuss algorithms that learn and output functions one needs to define how these functions are represented. Informally, a representation for a concept class C is a way to describe concepts from C that defines a procedure to evaluate a concept in C on any input. For example, one can represent a conjunction of input variables by listing the variables in the conjunction. More formally, a representation class can be defined as follows.

Definition 1 A *representation class* $\mathcal{F}$ is a pair $(L, \mathcal{R})$ where

- L is a language over some fixed finite alphabet (e. g. $\{0,1\}$);
- $\mathcal{R}$ is an algorithm that for $\sigma \in L$, on input $(\sigma, 1^n)$ returns a Boolean circuit over $\{0,1\}^n$.

In the context of efficient learning, only efficient representations are considered, or, representations for which $\mathcal{R}$ is a polynomial-time algorithm. The concept class represented by $\mathcal{F}$ is set of functions over $\{0,1\}^n$ defined by the circuits in $\{\mathcal{R}(\sigma, 1^n) \mid \sigma \in L\}$. For most of the representations discussed in the context of learning it is straightforward to construct a language L and the corresponding translating function $\mathcal{R}$, and therefore they are not specified explicitly.

Associated with each representation is the complexity of describing a Boolean function using this representation. More formally, for a Boolean function $f \in C$, $\mathcal{F}\texttt{-size}(f)$ is the length of the shortest way to represent f using $\mathcal{F}$, or $\min\{|\sigma| \mid \sigma \in L,\ \mathcal{R}(\sigma, 1^n) \equiv f\}$.

In Valiant's PAC model of learning, for a function f and a distribution $\mathcal{D}$ over X, an *example oracle* $\mathrm{EX}(f, \mathcal{D})$ is an oracle that, when invoked, returns an example

$\langle x, f(x) \rangle$, where x is chosen randomly with respect to $\mathcal{D}$, independently of any previous examples. For $\epsilon \geq 0$, a function g ϵ-approximates a function f with respect to distribution $\mathcal{D}$ if $\mathbf{Pr}_{\mathcal{D}}[f(x) \neq g(x)] \leq \epsilon$.

Definition 2 A representation class $\mathcal{F}$ is *PAC learnable* by representation class $\mathcal{H}$ if there exist an algorithm that for every $\epsilon > 0$, $\delta > 0$, n, $f \in \mathcal{F}$, and distribution $\mathcal{D}$ over X, given ϵ, δ, and access to $\text{EX}(f, \mathcal{D})$, runs in time polynomial in $n, s = \mathcal{F}\text{-}\mathtt{size}(c), 1/\epsilon$ and $1/\delta$, and outputs, with probability at least $1 - \delta$, a hypothesis $h \in \mathcal{H}$ that ϵ-approximates f.

A DNF expression is defined as an OR of ANDs of literals, where a *literal* is a possibly negated input variable. The ANDs of a DNF formula are referred to as its *terms*. Let DNF(k) denote the representation class of k-term DNF expressions. Similarly a CNF expression is an OR of ANDs of literals. Let k-CNF denote the representation class of CNF expressions with each AND having at most k literals.

For a real-valued vector $c \in \mathbb{R}^n$ and $\theta \in \mathbb{R}$, a *linear threshold function* (also called a *halfspace*) $T_{c,\theta}(x)$ is the function that equals 1 if and only if $\sum_{i \leq n} c_i x_i \geq \theta$. The representation class of Boolean threshold functions consists of all linear threshold functions with $c \in \{0,1\}^n$ and θ an integer.

Key Results

Theorem 3 ([16]) *For every $k \geq 2$, the representation class of* DNF(k) *is not properly learnable unless* RP = NP.

More specifically, Pitt and Valiant show that learning DNF(k) by DNF(ℓ) is at least as hard as coloring a k-colorable graph using ℓ colors. For the case $k = 2$ they obtain the result by reducing from Set Splitting (see [8] for details on the problems). Theorem 3 is in sharp contrast with the fact that DNF(k) is learnable by k-CNF [17].

Theorem 4 ([16]) *The representation class of Boolean threshold functions is not properly learnable unless* RP = NP.

This result is obtained via a reduction from the NP-complete Zero-One Integer Programming problem (see [8](p. 245) for details on the problem). The result is contrasted by the fact that general linear thresholds are properly learnable [4].

These results show that using a specific representation of hypotheses forces the learning algorithm to solve a combinatorial problem that can be NP-hard. In most machine learning applications it is not important which representation of hypotheses is used as long as the value of the unknown function is predicted correctly. Therefore learning in the PAC model is now defined without any restrictions on the output hypothesis (other than it being efficiently evaluatable). Hardness results in this setting are usually based on cryptographic assumptions (*cf.* [14]).

Hardness results for proper learning based on assumption NP $\neq$ RP are now known for several other representation classes and for other variants and extensions of the PAC learning model. Blum and Rivest show that for any $k \geq 3$, unions of k halfspaces are not properly learnable [3]. Hancock et al. prove that decision trees (*cf.* [15] for the definition of this representation) are not learnable by decision trees of somewhat larger size [10]. This result was strengthened by Alekhnovich et al. who also prove that intersections of two halfspaces are not learnable by intersections of k halfspaces for any constant k, general DNF expressions are not learnable by unions of halfspaces (and in particular are not properly learnable), and k-*juntas* are not properly learnable [1]. Feldman shows that DNF expressions are NP-hard to learn properly even if *membership queries*, or the ability to query the unknown function at any point, are allowed [6]. No efficient algorithms or hardness results are known for any of the above learning problems if no restriction is placed on the representation of hypotheses.

The choice of representation is very important even in powerful learning models. Feldman proved that n^c-term DNF are not properly learnable for any constant c even when the distribution of examples is assumed to be uniform and membership queries are available [6]. This contrasts with Jackson's celebrated algorithm for learning DNF in this setting [12], which is not proper.

In the *agnostic learning* model of Haussler [11] and Kearns et al. [13] even the representation classes of conjunctions, halfspaces, and parity functions are NP-hard to learn properly (*cf.* [2,7,9] and references therein). Here again the status of these problems in the representation-independent setting is largely unknown.

Applications

A large number of practical algorithms use representations for which hardness results are known (most notably decision trees, halfspaces, and neural networks). Hardness of learning $\mathcal{F}$ by $\mathcal{H}$ implies that an algorithm that uses $\mathcal{H}$ to represent its hypotheses will not be able to learn $\mathcal{F}$ in the PAC sense. Therefore such hardness results elucidate the limitations of algorithms used in practice. In particular, the reduction from an NP-hard problem used to prove the hardness of learning $\mathcal{F}$ by $\mathcal{H}$ can be used to generate hard instances of the learning problem.

Open Problems

A number of problems related to proper learning in the PAC model and its extensions are open. Almost all hardness of proper learning results are for learning with respect to unrestricted distributions. For most of the problems mentioned in Sect. "Key Results" it is unknown whether the result is true if the distribution is restricted to belong to some natural class of distributions (e. g. product distributions). It is unknown whether decision trees are learnable properly in the PAC model or in the PAC model with membership queries. This question is open even in the PAC model restricted to the uniform distribution only. Note that decision trees are learnable (non-properly) if membership queries are available [5] and are learnable properly in time $O(n^{\log s})$, where s is the number of leaves in the decision tree [1].

An even more interesting direction of research would be to obtain hardness results for learning by richer representations classes, such as AC^0 circuits, classes of neural networks and, ultimately, unrestricted circuits.

Cross References

► Cryptographic Hardness of Learning
► Graph Coloring
► Learning DNF Formulas
► PAC Learning

Recommended Reading

1. Alekhnovich, M., Braverman, M., Feldman, V., Klivans, A., Pitassi, T.: Learnability and automizability. In: Proceeding of FOCS, pp. 621–630 (2004)
2. Ben-David, S., Eiron, N., Long, P. M.: On the difficulty of approximately maximizing agreements. In: Proceedings of COLT, pp. 266–274 (2000)
3. Blum, A.L., Rivest, R.L.: Training a 3-node neural network is NP-complete. Neural Netw. **5**(1), 117–127 (1992)
4. Blumer, A., Ehrenfeucht, A., Haussler, D., Warmuth, M.: Learnability and the Vapnik-Chervonenkis dimension. J. ACM **36**(4), 929–965 (1989)
5. Bshouty, N.: Exact learning via the monotone theory. Inf. Comput. **123**(1), 146–153 (1995)
6. Feldman, V.: Hardness of Approximate Two-level Logic Minimization and PAC Learning with Membership Queries. In: Proceedings of STOC, pp. 363–372 (2006)
7. Feldman, V.: Optimal hardness results for maximizing agreements with monomials. In: Proceedings of Conference on Computational Complexity (CCC), pp. 226–236 (2006)
8. Garey, M., Johnson, D.S.: Computers and Intractability. W. H. Freeman, San Francisco (1979)
9. Guruswami, V., Raghavendra, P.: Hardness of Learning Halfspaces with Noise. In: Proceedings of FOCS, pp. 543–552 (2006)
10. Hancock, T., Jiang, T., Li, M., Tromp, J.: Lower bounds on learning decision lists and trees. In: 12th Annual Symposium on Theoretical Aspects of Computer Science, pp. 527–538 (1995)
11. Haussler, D.: Decision theoretic generalizations of the PAC model for neural net and other learning applications. Inf. Comput. **100**(1), 78–150 (1992)
12. Jackson, J.: An efficient membership-query algorithm for learning DNF with respect to the uniform distribution. J. Comput. Syst. Sci. **55**, 414–440 (1997)
13. Kearns, M., Schapire, R., Sellie, L.: Toward efficient agnostic learning. Mach. Learn. **17**(2–3), 115–141 (1994)
14. Kearns, M., Valiant, L.: Cryptographic limitations on learning boolean formulae and finite automata. J. ACM **41**(1), 67–95 (1994)
15. Kearns, M., Vazirani, U.: An introduction to computational learning theory. MIT Press, Cambridge, MA (1994)
16. Pitt, L., Valiant, L.: Computational limitations on learning from examples. J. ACM **35**(4), 965–984 (1988)
17. Valiant, L.: A theory of the learnable. Commun. ACM **27**(11), 1134–1142 (1984)

High Performance Algorithm Engineering for Large-scale Problems

2005; Bader

DAVID A. BADER
College of Computing, Georgia Institute of Technology, Atlanta, GA, USA

Keywords and Synonyms

Experimental algorithmics

Problem Definition

Algorithm engineering refers to the process required to transform a pencil-and-paper algorithm into a robust, efficient, well tested, and easily usable implementation. Thus it encompasses a number of topics, from modeling cache behavior to the principles of good software engineering; its main focus, however, is experimentation. In that sense, it may be viewed as a recent outgrowth of *Experimental Algorithmics* [14], which is specifically devoted to the development of methods, tools, and practices for assessing and refining algorithms through experimentation. The *ACM Journal of Experimental Algorithmics (JEA)*, at URL `www.jea.acm.org`, is devoted to this area.

High-performance algorithm engineering [2] focuses on one of the many facets of algorithm engineering: speed. The high-performance aspect does not immediately imply parallelism; in fact, in any highly parallel task, most of the impact of high-performance algorithm engineering tends to come from refining the serial part of the code.

The term *algorithm engineering* was first used with specificity in 1997, with the organization of the first *Workshop on Algorithm Engineering (WAE 97)*. Since then, this workshop has taken place every summer in Europe. The 1998 *Workshop on Algorithms and Experiments (ALEX98)* was held in Italy and provided a discussion forum for researchers and practitioners interested in the design, analyzes and experimental testing of exact and heuristic algorithms. A sibling workshop was started in the Unites States in 1999, the *Workshop on Algorithm Engineering and Experiments (ALENEX99)*, which has taken place every winter, colocated with the *ACM/SIAM Symposium on Discrete Algorithms (SODA)*.

Key Results

Parallel computing has two closely related main uses. First, with more memory and storage resources than available on a single workstation, a parallel computer can solve correspondingly larger instances of the same problems. This increase in size can translate into running higher-fidelity simulations, handling higher volumes of information in data-intensive applications, and answering larger numbers of queries and datamining requests in corporate databases. Secondly, with more processors and larger aggregate memory subsystems than available on a single workstation, a parallel computer can often solve problems faster. This increase in speed can also translate into all of the advantages listed above, but perhaps its crucial advantage is in turnaround time. When the computation is part of a real-time system, such as weather forecasting, financial investment decision-making, or tracking and guidance systems, turnaround time is obviously the critical issue. A less obvious benefit of shortened turnaround time is higher-quality work: when a computational experiment takes less than an hour, the researcher can afford the luxury of exploration—running several different scenarios in order to gain a better understanding of the phenomena being studied.

In algorithm engineering, the aim is to present repeatable results through experiments that apply to a broader class of computers than the specific make of computer system used during the experiment. For sequential computing, empirical results are often fairly machine-independent. While machine characteristics such as word size, cache and main memory sizes, and processor and bus speeds differ, comparisons across different uniprocessor machines show the same trends. In particular, the number of memory accesses and processor operations remains fairly constant (or within a small constant factor). In high-performance algorithm engineering with parallel computers, on the other hand, this portability is usually absent: each machine and environment is its own special case. One obvious reason is major differences in hardware that affect the balance of communication and computation costs—a true shared-memory machine exhibits very different behavior from that of a cluster based on commodity networks.

Another reason is that the communication libraries and parallel programming environments (e. g., MPI [12], OpenMP [16], and High-Performance Fortran [10]), as well as the parallel algorithm packages (e. g., fast Fourier transforms using FFTW [6] or parallelized linear algebra routines in ScaLAPACK [4]), often exhibit differing performance on different types of parallel platforms. When multiple library packages exist for the same task, a user may observe different running times for each library version even on the same platform. Thus a running-time analysis should clearly separate the time spent in the user code from that spent in various library calls. Indeed, if particular library calls contribute significantly to the running time, the number of such calls and running time for each call should be recorded and used in the analysis, thereby helping library developers focus on the most cost-effective improvements. For example, in a simple message-passing program, one can characterize the work done by keeping track of sequential work, communication volume, and number of communications. A more general program using the collective communication routines of MPI could also count the number of calls to these routines. Several packages are available to instrument MPI codes in order to capture such data (e. g., MPICH's nupshot [8], Pablo [17], and Vampir [15]). The SKaMPI benchmark [18] allows running-time predictions based on such measurements even if the target machine is not available for program development. SKaMPI was designed for robustness, accuracy, portability, and efficiency; For example, SKaMPI adaptively controls how often measurements are repeated, adaptively refines message-length and step-width at "interesting" points, recovers from crashes, and automatically generates reports.

Applications

The following are several examples of algorithm engineering studies for high-performance and parallel computing.

1. Bader's prior publications (see [2] and http://www.cc.gatech.edu/~bader) contain many empirical studies of parallel algorithms for combinatorial problems like sorting, selection, graph algorithms, and image processing.
2. In a recent demonstration of the power of high-performance algorithm engineering, a million-fold speed-up was achieved through a combination of a 2,000-fold speedup in the serial execution of the code and a 512-

fold speedup due to parallelism (a speed-up, however, that will scale to any number of processors) [13]. (In a further demonstration of algorithm engineering, additional refinements in the search and bounding strategies have added another speedup to the serial part of about 1,000, for an overall speedup in excess of 2 billion)

3. JáJá and Helman conducted empirical studies for prefix computations, sorting, and list-ranking, on symmetric multiprocessors. The sorting research (see [9]) extends Vitter's external Parallel Disk Model to the internal memory hierarchy of SMPs and uses this new computational model to analyze a general-purpose sample sort that operates efficiently in shared-memory. The performance evaluation uses 9 well-defined benchmarks. The benchmarks include input distributions commonly used for sorting benchmarks (such as keys selected uniformly and at random), but also benchmarks designed to challenge the implementation through load imbalance and memory contention and to circumvent algorithmic design choices based on specific input properties (such as data distribution, presence of duplicate keys, pre-sorted inputs, etc.).
4. In [3] Blelloch et al. compare through analysis and implementation three sorting algorithms on the Thinking Machines CM-2. Despite the use of an outdated (and no longer available) platform, this paper is a gem and should be required reading for every parallel algorithm designer. In one of the first studies of its kind, the authors estimate running times of four of the machine's primitives, then analyze the steps of the three sorting algorithms in terms of these parameters. The experimental studies of the performance are normalized to provide clear comparison of how the algorithms scale with input size on a $32K$-processor CM-2.
5. Vitter et al. provide the canonical theoretic foundation for I/O-intensive experimental algorithmics using external parallel disks (e. g., see [1,19,20]). Examples from sorting, FFT, permuting, and matrix transposition problems are used to demonstrate the parallel disk model.
6. Juurlink and Wijshoff [11] perform one of the first detailed experimental accounts on the preciseness of several parallel computation models on five parallel platforms. The authors discuss the predictive capabilities of the models, compare the models to find out which allows for the design of the most efficient parallel algorithms, and experimentally compare the performance of algorithms designed with the model versus those designed with machine-specific characteristics in mind. The authors derive model parameters for each platform, analyses for a variety of algorithms (matrix multiplication, bitonic sort, sample sort, all-pairs shortest path), and detailed performance comparisons.
7. The LogP model of Culler et al. [5] provides a realistic model for designing parallel algorithms for message-passing platforms. Its use is demonstrated for a number of problems, including sorting.
8. Several research groups have performed extensive algorithm engineering for high-performance numerical computing. One of the most prominent efforts is that led by Dongarra for ScaLAPACK [4], a scalable linear algebra library for parallel computers. ScaLAPACK encapsulates much of the high-performance algorithm engineering with significant impact to its users who require efficient parallel versions of matrix–matrix linear algebra routines. New approaches for automatically tuning the sequential library (e. g., LAPACK) are now available as the ATLAS package [21].

Open Problems

All of the tools and techniques developed over the last several years for algorithm engineering are applicable to high-performance algorithm engineering. However, many of these tools need further refinement. For example, cache-efficient programming is a key to performance but it is not yet well understood, mainly because of complex machine-dependent issues like limited associativity, virtual address translation, and increasingly deep hierarchies of high-performance machines. A key question is whether one can find simple models as a basis for algorithm development. For example, cache-oblivious algorithms [7] are efficient at all levels of the memory hierarchy in theory, but so far only few work well in practice. As another example, profiling a running program offers serious challenges in a serial environment (any profiling tool affects the behavior of what is being observed), but these challenges pale in comparison with those arising in a parallel or distributed environment (for instance, measuring communication bottlenecks may require hardware assistance from the network switches or at least reprogramming them, which is sure to affect their behavior). Designing efficient and portable algorithms for commodity multicore and many-core processors is an open challenge.

Cross References

- ► Analyzing Cache Misses
- ► Cache-Oblivious B-Tree
- ► Cache-Oblivious Model
- ► Cache-Oblivious Sorting
- ► Engineering Algorithms for Computational Biology

▶ Engineering Algorithms for Large Network Applications
▶ Engineering Geometric Algorithms
▶ Experimental Methods for Algorithm Analysis
▶ External Sorting and Permuting
▶ Implementation Challenge for Shortest Paths
▶ Implementation Challenge for TSP Heuristics
▶ I/O-model
▶ Visualization Techniques for Algorithm Engineering

Recommended Reading

1. Aggarwal, A., Vitter, J.: The input/output complexity of sorting and related problems. Commun. ACM **31**, 1116–1127 (1988)
2. Bader, D.A., Moret, B.M.E., Sanders, P.: Algorithm engineering for parallel computation. In: Fleischer, R., Meineche-Schmidt, E., Moret, B.M.E. (ed) Experimental Algorithmics. Lecture Notes in Computer Science, vol. 2547, pp. 1–23. Springer, Berlin (2002)
3. Blelloch, G.E., Leiserson, C.E., Maggs, B.M., Plaxton, C.G., Smith, S.J., Zagha, M.: An experimental analysis of parallel sorting algorithms. Theor. Comput. Syst. **31**(2), 135–167 (1998)
4. Choi, J., Dongarra, J.J., Pozo, R., Walker, D.W.: ScaLAPACK: A scalable linear algebra library for distributed memory concurrent computers. In: The 4th Symp. the Frontiers of Massively Parallel Computations, pp. 120–127, McLean, VA (1992)
5. Culler, D.E., Karp, R.M., Patterson, D.A., Sahay, A., Schauser, K.E., Santos, E., Subramonian, R., von Eicken,T.: LogP: Towards a realistic model of parallel computation. In: 4th Symp. Principles and Practice of Parallel Programming, pp. 1–12. ACM SIGPLAN (1993)
6. Frigo, M., Johnson, S. G.: FFTW: An adaptive software architecture for the FFT. In: Proc. IEEE Int'l Conf. Acoustics, Speech, and Signal Processing, vol. 3, pp. 1381–1384, Seattle, WA (1998)
7. Frigo, M., Leiserson, C.E., Prokop, H., Ramachandran, S.: Cache-oblivious algorithms. In: Proc. 40th Ann. Symp. Foundations of Computer Science (FOCS-99), pp. 285–297, New York, NY, 1999. IEEE Press
8. Gropp, W., Lusk, E., Doss, N., Skjellum, A.: A high-performance, portable implementation of the MPI message passing interface standard. Technical report, Argonne National Laboratory, Argonne, IL, (1996) www.mcs.anl.gov/mpi/mpich/
9. Helman, D.R., JáJá, J.: Sorting on clusters of SMP's. In: Proc. 12th Int'l Parallel Processing Symp., pp. 1–7, Orlando, FL, March/April 1998
10. High Performance Fortran Forum. High Performance Fortran Language Specification, 1.0 edition, May 1993
11. Juurlink, B.H.H., Wijshoff, H.A.G.: A quantitative comparison of parallel computation models. ACM Trans. Comput. Syst. **13**(3), 271–318 (1998)
12. Message Passing Interface Forum. MPI: A message-passing interface standard. Technical report, University of Tennessee, Knoxville, TN, June 1995. Version 1.1
13. Moret, B.M.E., Bader, D.A., Warnow, T.: High-performance algorithm engineering for computational phylogenetics. J. Supercomput. **22**, 99–111 (2002) Special issue on the best papers from ICCS'01
14. Moret, B.M.E., Shapiro, H.D.: Algorithms and experiments: The new (and old) methodology. J. Univers. Comput. Sci. **7**(5), 434–446 (2001)
15. Nagel, W.E., Arnold, A., Weber, M., Hoppe, H.C., Solchenbach, K.: VAMPIR: visualization and analysis of MPI resources. Supercomputer 63. **12**(1), 69–80 (1996)
16. OpenMP Architecture Review Board. OpenMP: A proposed industry standard API for shared memory programming. www.openmp.org, October 1997
17. Reed, D.A., Aydt, R.A., Noe, R.J., Roth, P.C., Shields, K.A., Schwartz, B., Tavera, L.F.: Scalable performance analysis: The Pablo performance analysis environment. In: Skjellum, A., (ed) Proc. Scalable Parallel Libraries Conf., pp. 104–113, Mississippi State University, October 1993. IEEE Computer Society Press
18. Reussner, R., Sanders, P., Träff, J.: SKaMPI: A comprehensive benchmark for public benchmarking of MPI. Scientific Programming, 2001. accepted, conference version with Prechelt, L., Müller, M. In: Proc. EuroPVM/MPI (1998)
19. Vitter, J. S., Shriver, E.A.M.: Algorithms for parallel memory. I: Two-level memories. Algorithmica. **12**(2/3), 110–147 (1994)
20. Vitter, J. S., Shriver, E.A.M.: Algorithms for parallel memory II: Hierarchical multilevel memories. Algorithmica **12**(2/3), 148–169 (1994)
21. Whaley, R., Dongarra, J.: Automatically tuned linear algebra software (ATLAS). In: Proc. Supercomputing 98, Orlando, FL, November 1998. www.netlib.org/utk/people/JackDongarra/PAPERS/atlas-sc98.ps

Hitting Set

▶ Greedy Set-Cover Algorithms
▶ Set Cover with Almost Consecutive Ones

Hospitals/Residents Problem

1962; Gale, Shapley

David F. Manlove
Department of Computing Science,
University of Glasgow, Glasgow, UK

Keywords and Synonyms

College admissions problem; University admissions problem; Stable admissions problem; Stable assignment problem; Stable *b*-matching problem

Problem Definition

An instance I of the Hospitals/Residents problem (HR) [5,6,14] involves a set $R = \{r_1, \ldots, r_n\}$ of *residents* and a set $H = \{h_1, \ldots, h_m\}$ of *hospitals*. Each hospital $h_j \in H$ has a positive integral *capacity*, denoted by c_j. Also, each resident $r_i \in R$ has a *preference list* in which he ranks in strict order a subset of H. A pair $(r_i, h_j) \in R \times H$ is said to be *acceptable* if h_j appears in r_i's preference list; in this case r_i is said to *find* h_j *acceptable*. Similarly each hospital $h_j \in H$ has a preference list in which it ranks in strict order those residents who find h_j acceptable. Given any three agents $x, y, z \in R \cup H$, x is said to *prefer* y to z if x finds each of y and z acceptable, and y precedes z on x's preference list. Let $C = \sum_{h_j \in H} c_j$.

Let A denote the set of acceptable pairs in I, and let $L = |A|$. An *assignment* M is a subset of A. If $(r_i, h_j) \in M$, r_i is said to be *assigned to* h_j, and h_j is *assigned* r_i. For each $q \in R \cup H$, the set of assignees of q in M is denoted by $M(q)$. If $r_i \in R$ and $M(r_i) = \emptyset$, r_i is said to be *unassigned*, otherwise r_i is *assigned*. Similarly, any hospital $h_j \in H$ is *under-subscribed*, *full* or *over-subscribed* according as $|M(h_j)|$ is less than, equal to, or greater than c_j, respectively.

A *matching* M is an assignment such that $|M(r_i)| \leq 1$ for each $r_i \in R$ and $|M(h_j)| \leq c_j$ for each $h_j \in H$ (i. e., no resident is assigned to an unacceptable hospital, each resident is assigned to at most one hospital, and no hospital is over-subscribed). For notational convenience, given a matching M and a resident $r_i \in R$ such that $M(r_i) \neq \emptyset$, where there is no ambiguity the notation $M(r_i)$ is also used to refer to the single member of $M(r_i)$.

A pair $(r_i, h_j) \in A \backslash M$ *blocks* a matching M, or is a *blocking pair* for M, if the following conditions are satisfied relative to M:

1. r_i is unassigned or prefers h_j to $M(r_i)$;
2. h_j is under-subscribed or prefers r_i to at least one member of $M(h_j)$ (or both).

A matching M is said to be *stable* if it admits no blocking pair. Given an instance I of HR, the problem is to find a stable matching in I.

Key Results

HR was first defined by Gale and Shapley [5] under the name "College Admissions Problem". In their seminal paper, the authors' primary consideration is the classical Stable Marriage problem (SM; see ▶ Stable Marriage and ▶ Optimal Stable Marriage), which is a special case of HR in which $n = m$, $A = R \times H$, and $c_j = 1$ for all $h_j \in H$ – in this case, the residents and hospitals are more commonly referred to as the men and women respectively. Gale and Shapley show that every instance I of HR admits at least one stable matching. Their proof of this result is constructive, i. e., an algorithm for finding a stable matching in I is described. This algorithm has become known as the *Gale/Shapley algorithm*.

An extended version of the Gale/Shapley algorithm for HR is shown in Fig. 1. The algorithm involves a sequence of *apply* and *delete* operations. At each iteration of the while loop, some unassigned resident r_i with a non-empty preference list applies to the first hospital h_j on his list, and becomes provisionally assigned to h_j (this assignment could subsequently be broken). If h_j becomes over-subscribed as a result of this assignment, then h_j rejects its worst assigned resident r_k. Next, if h_j is full (irrespective of whether h_j was over-subscribed earlier in the same loop iteration), then for each resident r_l that h_j finds less de-

```
$M := \emptyset$;
**while** (some resident $r_i$ is unassigned **and** $r_i$ has a non-empty list) {
    $h_j$ := first hospital on $r_i$'s list;
    /* $r_i$ applies to $h_j$ */
    $M := M \cup \{(r_i, h_j)\}$;
    **if** ($h_j$ is over-subscribed) {
        $r_k$ := worst resident in $M(h_j)$ according to $h_j$'s list;
        $M := M \backslash \{(r_k, h_j)\}$;
    }
    **if** ($h_j$ is full) {
        $r_k$ := worst resident in $M(h_j)$ according to $h_j$'s list;
        **for** (each successor $r_l$ of $r_k$ on $h_j$'s list)
            delete the pair $(r_l, h_j)$;
    }
}
```

Hospitals/Residents Problem, Figure 1
Gale/Shapley algorithm for HR

sirable than its worst resident r_k, the algorithm *deletes the pair* (r_l, h_j), which comprises deleting h_j from r_l's preference list and vice versa.

Given that the above algorithm involves residents applying to hospitals, it has become known as the *Resident-oriented* Gale/Shapley algorithm, or RGS algorithm for short [6, Sect. 1.6.3]. The RGS algorithm terminates with a stable matching, given an instance of HR [5,6, Theorem 1.6.2]. Using a suitable choice of data structures (extending those described in [6, Sect. 1.2.3]), the RGS algorithm can be implemented to run in $O(L)$ time. This algorithm produces the stable matching that is simultaneously best-possible for all residents [5,6, Theorem 1.6.2]. These observations may be summarized as follows:

Theorem 1 *Given an instance of HR, the RGS algorithm constructs, in O(L) time, the unique stable matching in which each assigned resident obtains the best hospital that he could obtain in any stable matching, whilst each unassigned resident is unassigned in every stable matching.*

A counterpart of the RGS algorithm, known as the *Hospital-oriented* Gale/Shapley algorithm, or HGS algorithm for short [6, Sect. 1.6.2], gives the unique stable matching that similarly satisfies an optimality property for the hospitals [6, Theorem 1.6.1].

Although there may be many stable matchings for a given instance I of HR, some key structural properties hold regarding unassigned residents and under-subscribed hospitals with respect to all stable matchings in I, as follows.

Theorem 2 *For a given instance of HR,*
- *the same residents are assigned in all stable matchings;*
- *each hospital is assigned the same number of residents in all stable matchings;*
- *any hospital that is under-subscribed in one stable matching is assigned exactly the same set of residents in all stable matchings.*

These results are collectively known as the "Rural Hospitals Theorem" (see [6, Sect. 1.6.4] for further details). Furthermore, the set of stable matchings in I forms a distributive lattice under a natural dominance relation [6, Sect. 1.6.5].

Applications

Practical applications of HR are widespread, most notably arising in the context of centralized automated matching schemes that assign applicants to posts (for example medical students to hospitals, school-leavers to universities, and primary school pupils to secondary schools). Perhaps the best-known example of such a scheme is the National Resident Matching Program (NRMP) in the US [16], which annually assigns around 31,000 graduating medical students (known as residents) to their first hospital posts, taking into account the preferences of residents over hospitals and vice versa, and the hospital capacities. Counterparts of the NRMP are in existence in other countries, including Canada [17], Scotland [18] and Japan [19]. These matching schemes essentially employ extensions of the RGS algorithm for HR.

Centralized matching schemes based largely on HR also occur in other practical contexts, such as school placement in New York [1], university faculty recruitment in France [3] and university admission in Spain [12].

Extensions of HR

One key extension of HR that has considerable practical importance arises when an instance may involve a set of couples, each of which submits a joint preference list over pairs of hospitals (typically in order that the members of a given couple can be located geographically close to one another, for example). The extension of HR in which couples may be involved is denoted by HRC; the stability definition in HRC is a natural extension of that in HR (see [6, Sect. 1.6.6] for a formal definition of HRC). It is known that an instance of HRC need not admit a stable matching (see [6, Section 1.6.6] and [14, Sect. 5.4.3]). Moreover, the problem of deciding whether an HRC instance admits a stable matching is NP-complete [13].

HR may be regarded as a many-one generalization of SM. A further generalization of SM is to a many-many stable matching problem, in which both residents and hospitals may be multiply assigned subject to capacity constraints. In this case, residents and hospitals are more commonly referred to as workers and firms respectively. There are two basic variations of the many-many stable matching problem according to whether (i) workers rank acceptable firms in order of preference and vice versa, or (ii) workers rank acceptable *subsets* of firms in order of preference and vice versa. Previous work relating to both models is surveyed in [4].

Other variants of HR may be obtained if preference lists include ties. This extension is again important from a practical perspective, since it may be unrealistic to expect a popular hospital to rank a large number of applicants in strict order, particularly if it is indifferent among groups of applicants. The extension of HR in which preference lists may include ties is denoted by HRT. In this context three natural stability definitions arise, so-called *weak stability*, *strong stability* and *super-stability* (see [8] for formal definitions of these concepts). Given an instance I of HRT, it is known that weakly stable matchings may have different sizes, and the problem of finding a maximum cardinality weakly stable matching is NP-hard (see ▶ Stable Marriage with Ties and Incomplete Lists for further details). On the other hand, in contrast to the case for weak stability, a super-stable matching in I need not exist, though there is an $O(L)$ algorithm to find a such a matching if one does [7]. Analogous results hold in the case of strong stability – in this case an $O(L^2)$ algorithm [8] was improved by an $O(CL)$ algorithm [10] and extended to the many-many case [11]. Furthermore, counterparts of the Rural Hospitals Theorem hold for HRT under each of the super-stability and strong stability criteria [7,15].

A further generalization of HR arises when each hospital may be split into several departments, where each department has a capacity, and residents rank individual departments in order of preference. This variant is modeled by the Student-Project Allocation problem [2]. Finally, the Stable Fixtures problem [9] is a non-bipartite extension of HR in which there is a single set of agents, each of whom has a capacity and ranks a subset of the others in order of preference.

Open Problems

Several approximation algorithms for finding a maximum cardinality weakly stable matching have been formulated, given an instance of HRT where each hospital has capacity 1 (see ▶ Stable Marriage with Ties and Incomplete Lists for further details). It remains open to extend these algorithms or to formulate effective heuristics for the case of HRT with arbitrary capacities. This problem is particularly relevant from the practical perspective, since as already noted in Sect. "Applications", hospitals may wish to include ties in their preference lists. In this case weak stability is the most commonly-used stability criterion, due to the guaranteed existence of such a matching. Attempting to match as many residents as possible motivates the search for large weakly stable matchings.

URL to Code

Ada implementations of the RGS and HGS algorithms for HR may be found via the following URL: http://www.dcs.gla.ac.uk/research/algorithms/stable.

Cross References

▶ Optimal Stable Marriage
▶ Ranked Matching
▶ Stable Marriage
▶ Stable Marriage and Discrete Convex Analysis
▶ Stable Marriage with Ties and Incomplete Lists
▶ Stable Partition Problem

Recommended Reading

1. Abdulkadiroğlu, A., Pathak, P.A., Roth, A.E.: The New York City high school match. Am. Economic. Rev. **95**(2), 364–367 (2006)
2. Abraham, D.J., Irving, R.W., Manlove, D.F.: Two algorithms for the Student-Project allocation problem. J. Discret. Algorithms **5**(1), 73–90 (2007)
3. Baïou, M., Balinski, M.: Student admissions and faculty recruitment. Theor. Comput. Sci. **322**(2), 245–265 (2004)
4. Bansal, V., Agrawal, A., Malhotra, V.S.: Stable marriages with multiple partners: efficient search for an optimal solution. In: Proceedings of ICALP '03: the 30th International Colloquium on Automata, Languages and Programming. Lecture Notes in Computer Science, vol. 2719, pp. 527–542. Springer, Berlin (2003)
5. Gale, D., Shapley, L.S.: College admissions and the stability of marriage. Am. Math. Month. **69**, 9–15 (1962)
6. Gusfield, D., Irving, R.W.: The Stable Marriage Problem: Structure and Algorithms. MIT Press, Cambridge (1989)
7. Irving, R.W., Manlove, D.F., Scott, S.: The Hospitals/Residents problem with Ties. In: Proceedings of SWAT 2000: the 7th Scandinavian Workshop on Algorithm Theory. Lecture Notes in Computer Science, vol. 1851, pp. 259–271. Springer, Berlin (2000)
8. Irving, R.W., Manlove, D.F., Scott, S.: Strong stability in the Hospitals/Residents problem. In: Proceedings of STACS 2003: the 20th Annual Symposium on Theoretical Aspects of Computer Science. Lecture Notes in Computer Science, vol. 2607, pp. 439–450. Springer, Berlin (2003)
9. Irving, R.W., Scott, S.: The stable fixtures problem – a many-to-many extension of stable roommates. Discret. Appl. Math. **155**, 2118–2129 (2007)
10. Kavitha, T., Mehlhorn, K., Michail, D., Paluch, K.: Strongly stable matchings in time $O(nm)$ and extension to the Hospitals-Residents problem. In: Proceedings of STACS 2004: the 21st International Symposium on Theoretical Aspects of Computer Science. Lecture Notes in Computer Science, vol. 2996, pp. 222–233. Springer, Berlin (2004)
11. Malhotra, V.S.: On the stability of multiple partner stable marriages with ties. In: Proceedings of ESA '04: the 12th Annual European Symposium on Algorithms. Lecture Notes in Computer Science, vol. 3221, pp. 508–519. Springer, Berlin (2004)
12. Romero-Medina, A.: Implementation of stable solutions in a restricted matching market. Rev. Economic. Des. **3**(2), 137–147 (1998)
13. Ronn, E.: NP-complete stable matching problems. J. Algorithms **11**, 285–304 (1990)
14. Roth, A.E., Sotomayor, M.A.O.: Two-sided matching: a study in game-theoretic modeling and analysis. Econometric Society

Monographs, vol. 18. Cambridge University Press, Cambridge, UK (1990)
15. Scott, S.: A study of stable marriage problems with ties. Ph. D. thesis, University of Glasgow, Dept. Comput. Sci. (2005)
16. http://www.nrmp.org/ (National Resident Matching Program website)
17. http://www.carms.ca (Canadian Resident Matching Service website)
18. http://www.nes.scot.nhs.uk/sfas/ (Scottish Foundation Allocation Scheme website)
19. http://www.jrmp.jp (Japan Resident Matching Program website)

I

Implementation Challenge for Shortest Paths

2006; Demetrescu, Goldberg, Johnson

CAMIL DEMETRESCU[1], ANDREW V. GOLDBERG[2], DAVID S. JOHNSON[3]
[1] Department of Information and Computer Systems, University of Roma, Rome, Italy
[2] Microsoft Research – Silicon Valley, Mountain View, CA, USA
[3] Algorithms and Optimization Research Dept., AT&T Labs, Florham Park, NJ, USA

Keywords and Synonyms

Test sets and experimental evaluation of computer programs for solving shortest path problems; DIMACS

Problem Definition

DIMACS Implementation Challenges (http://dimacs.rutgers.edu/Challenges/) are scientific events devoted to assessing the practical performance of algorithms in experimental settings, fostering effective technology transfer and establishing common benchmarks for fundamental computing problems. They are organized by DIMACS, the Center for Discrete Mathematics and Theoretical Computer Science. One of the main goals of DIMACS Implementation Challenges is to address questions of determining realistic algorithm performance where worst case analysis is overly pessimistic and probabilistic models are too unrealistic: experimentation can provide guides to realistic algorithm performance where analysis fails. Experimentation also brings algorithmic questions closer to the original problems that motivated theoretical work. It also tests many assumptions about implementation methods and data structures. It provides an opportunity to develop and test problem instances, instance generators, and other methods of testing and comparing performance of algorithms. And it is a step in technology transfer by providing leading edge implementations of algorithms for others to adapt.

The first Challenge was held in 1990–1991 and was devoted to *Network flows and Matching*. Other addressed problems included: *Maximum Clique, Graph Coloring, and Satisfiability* (1992–1993), *Parallel Algorithms for Combinatorial Problems* (1993–1994), *Fragment Assembly and Genome Rearrangements* (1994–1995), *Priority Queues, Dictionaries, and Multi-Dimensional Point Sets* (1995–1996), *Near Neighbor Searches* (1998–1999), *Semidefinite and Related Optimization Problems* (1999–2000), and *The Traveling Salesman Problem* (2000–2001).

This entry addresses the goals and the results of the *9th DIMACS Implementation Challenge*, held in 2005–2006 and focused on *Shortest Path* problems.

The 9th DIMACS Implementation Challenge: The Shortest Path Problem

Shortest path problems are among the most fundamental combinatorial optimization problems with many applications, both direct and as subroutines in other combinatorial optimization algorithms. Algorithms for these problems have been studied since the 1950's and still remain an active area of research.

One goal of this Challenge was to create a reproducible picture of the state of the art in the area of shortest path algorithms, identifying a standard set of benchmark instances and generators, as well as benchmark implementations of well-known shortest path algorithms. Another goal was to enable current researchers to compare their codes with each other, in hopes of identifying the more effective of the recent algorithmic innovations that have been proposed.

Challenge participants studied the following variants of the shortest paths problem:

- *Point to point shortest paths* [4,5,6,9,10,11,14]: the problem consists of answering multiple online queries about the shortest paths between pairs of vertices and/or their lengths. The most efficient solutions for this problem preprocess the graph to create a data structure that facilitates answering queries quickly.

- *External-memory shortest paths* [2]: the problem consists of finding shortest paths in a graph whose size is too large to fit in internal memory. The problem actually addressed in the Challenge was single-source shortest paths in undirected graphs with unit edge weights.
- *Parallel shortest paths* [8,12]: the problem consists of computing shortest paths using multiple processors, with the goal of achieving good speedups over traditional sequential implementations. The problem actually addressed in the Challenge was single-source shortest paths.
- *K-shortest paths* [13,15]: the problem consists of ranking paths between a pair of vertices by non decreasing order of their length.
- *Regular-language constrained shortest paths:* [3] the problem consists of a generalization of shortest path problems where paths must satisfy certain constraints specified by a regular language. The problems studied in the context of the Challenge were single-source and point-to-point shortest paths, with applications ranging from transportation science to databases.

The Challenge culminated in a Workshop held at the DIMACS Center at Rutgers University, Piscataway, New Jersey on November 13–14, 2006. Papers presented at the conference are available at the URL: http://www.dis.uniroma1.it/~challenge9/papers.shtml. Selected contributions are expected to appear in a book published by the American Mathematical Society in the DIMACS Book Series.

Key Results

The main results of the 9th DIMACS Implementation Challenge include:

- Definition of common file formats for several variants of the shortest path problem, both static and dynamic. These include an extension of the famous DIMACS graph file format used by several algorithmic software libraries. Formats are described at the URL: http://www.dis.uniroma1.it/~challenge9/formats.shtml.
- Definition of a common set of core input instances for evaluating shortest path algorithms.
- Definition of benchmark codes for shortest path problems.
- Experimental evaluation of state-of-the-art implementations of shortest path codes on the core input families.
- A discussion of directions for further research in the area of shortest paths, identifying problems critical in real-world applications for which efficient solutions still remain unknown.

The chief information venue about the 9th DIMACS Implementation Challenge is the website http://www.dis.uniroma1.it/~challenge9.

Applications

Shortest path problems arise naturally in a remarkable number of applications. A limited list includes transportation planning, network optimization, packet routing, image segmentation, speech recognition, document formatting, robotics, compilers, traffic information systems, and dataflow analysis. It also appears as a subproblem of several other combinatorial optimization problems such as network flows. A comprehensive discussion of applications of shortest path problems appears in [1].

Open Problems

There are several open questions related to shortest path problems, both theoretical and practical. One of the most prominent discussed at the 9th DIMACS Challenge Workshop is modeling traffic fluctuations in point-to-point shortest paths. The current fastest implementations preprocess the input graph to answer point-to-point queries efficiently, and this operation may take hours on graphs arising in large-scale road map navigation systems. A change in the traffic conditions may require rescanning the whole graph several times. Currently, no efficient technique is known for updating the preprocessing information without rebuilding it from scratch. This would have a major impact on the performance of routing software.

Data Sets

The collection of benchmark inputs of the 9th DIMACS Implementation Challenge includes both synthetic and real-world data. All graphs are strongly connected. Synthetic graphs include random graphs, grids, graphs embedded on a torus, and graphs with small-world properties. Real-world inputs consist of graphs representing the road networks of Europe and USA. Europe graphs are provided by courtesy of the PTV company, Karlsruhe, Germany, subject to signing a (no-cost) license agreement. They include the road networks of 17 European countries: AUT, BEL, CHE, CZE, DEU, DNK, ESP, FIN, FRA, GBR, IRL, ITA, LUX, NDL, NOR, PRT, SWE, with a total of about 19 million nodes and 23 million edges. USA graphs are derived from the *UA Census 2000 TIGER/Line*

Implementation Challenge for Shortest Paths, Table 1
USA Road Networks derived from the TIGER/Line collection

NAME	DESCRIPTION	NODES	ARCS	BOUNDING BOX LATITUDE (N)	BOUNDING BOX LONGITUDE (W)
USA	Full USA	23 947 347	58 333 344	–	–
CTR	Central USA	14 081 816	34 292 496	[25.0; 50.0]	[79.0; 100.0]
W	Western USA	6 262 104	15 248 146	[27.0; 50.0]	[100.0; 130.0]
E	Eastern USA	3 598 623	8 778 114	[24.0; 50.0]	[-∞; 79.0]
LKS	Great Lakes	2 758 119	6 885 658	[41.0; 50.0]	[74.0; 93.0]
CAL	California and Nevada	1 890 815	4 657 742	[32.5; 42.0]	[114.0; 125.0]
NE	Northeast USA	1 524 453	3 897 636	[39.5, 43.0]	[-∞; 76.0]
NW	Northwest USA	1 207 945	2 840 208	[42.0; 50.0]	[116.0; 126.0]
FLA	Florida	1 070 376	2 712 798	[24.0; 31.0]	[79; 87.5]
COL	Colorado	435 666	1 057 066	[37.0; 41.0]	[102.0; 109.0]
BAY	Bay Area	321 270	800 172	[37.0; 39.0]	[121; 123]
NY	New York City	264 346	733 846	[40.3; 41.3]	[73.5; 74.5]

Implementation Challenge for Shortest Paths, Table 2
Results of the Challenge competition on the USA graph (23.9 million nodes and 58.3 million arcs) with unit arc lengths. The benchmark ratio is the average query time divided by the time required to answer a query using the Challenge Dijkstra benchmark code on the same platform. Query times and node scans are average values per query over 1000 random queries

	PREPROCESSING		QUERY		
CODE	Time (minutes)	Space (MB)	Node scans	Time (ms)	Benchmark ratio
HH-based transit [14]	104	3664	n.a.	0.019	$4.78 \cdot 10^{-6}$
TRANSIT [4]	720	n.a.	n.a.	0.052	$10.77 \cdot 10^{-6}$
HH Star [6]	32	2662	1082	1.14	$287.32 \cdot 10^{-6}$
REAL(16,1) [9]	107	2435	823	1.42	$296.30 \cdot 10^{-6}$
HH with DistTab [6]	29	2101	1671	1.61	$405.77 \cdot 10^{-6}$
RE [9]	88	861	3065	2.78	$580.08 \cdot 10^{-6}$

Files produced by the Geography Division of the US Census Bureau, Washington, DC. The TIGER/Line collection is available at: http://www.census.gov/geo/www/tiger/tigerua/ua_tgr2k.html. The Challenge USA core family contains a graph representing the full USA road system with about 24 million nodes and 58 million edges, plus 11 subgraphs obtained by cutting it along different bounding boxes as shown in Table 1. Graphs in the collection include also node coordinates and are given in DIMACS format.

The benchmark input package also features query generators for the single-source and point-to-point shortest path problems. For the single-source version, sources are randomly chosen. For the point-to-point problem, both random and local queries are considered. Local queries of the form (s, t) are generated by randomly picking t among the nodes with rank in $[2^i, 2^{i+1})$ in the ordering in which nodes are scanned by Dijkstra's algorithm with source s, for any parameter i. Clearly, the smaller i is, the closer nodes s and t are in the graph. Local queries are important to test how the algorithms' performance is affected by the distance between query endpoints.

The core input families of the 9th DIMACS Implementation Challenge are available at the URL: http://www.dis.uniroma1.it/~challenge9/download.shtml.

Experimental Results

One of the main goals of the Challenge was to compare different techniques and algorithmic approaches. The most popular topic was the point-to-point shortest path problem, studied by six research groups in the context of the Challenge. For this problem, participants were additionally invited to join a competition aimed at assessing the performance and the robustness of different implementations. The competition consisted of preprocessing a version of the full USA graph of Table 1 with unit edge lengths and answering a sequence of 1,000 random distance queries. The details were announced on the first day of the workshop and the results were due on the second day. To compare experimental results by different participants on different platforms, each participant ran a Dijkstra benchmark code [7] on the USA graph to do machine calibration. The final ranking was made by considering each query time divided by the time required by the benchmark code on the same platform (benchmark ratio). Other performance measures taken into account were space usage and the average number of nodes scanned by query operations.

Six point-to-point implementations were run successfully on the USA graph defined for the competition. Among them, the fastest query time was achieved by the *HH-based transit* code [14]. Results are reported in Table 2. Codes *RE* and *REAL*(16, 1) [9] were not eligible for the competition, but used by the organizers as a proof that the problem is feasible. Some other codes were not able to deal with the size of the full USA graph, or incurred runtime errors.

Experimental results for other variants of the shortest paths problem are described in the papers presented at the Challenge Workshop.

URL to Code

Generators of problem families and benchmark solvers for shortest paths problems are available at the URL: http://www.dis.uniroma1.it/~challenge9/download.shtml.

Cross References

► Engineering Algorithms for Large Network Applications
► Experimental Methods for Algorithm Analysis
► High Performance Algorithm Engineering for Large-scale Problems
► Implementation Challenge for TSP Heuristics
► LEDA: a Library of Efficient Algorithms

Recommended Reading

1. Ahuja, R., Magnanti, T., Orlin, J.: Network Flows: Theory, Algorithms and Applications. Prentice Hall, Englewood Cliffs, NJ (1993)
2. Ajwani, D., Dementiev, U., Meyer, R., Osipov, V.: Breadth first search on massive graphs. In: 9th DIMACS Implementation Challenge Workshop: Shortest Paths, DIMACS Center, Piscataway, NJ, 13–14 Nov 2006
3. Barrett, C., Bissett, K., Holzer, M., Konjevod, G., Marathe, M., Wagner, D.: Implementations of routing algorithms for transportation networks. In: 9th DIMACS Implementation Challenge Workshop: Shortest Paths. DIMACS Center, Piscataway, NJ, 13–14 Nov 2006
4. Bast, H., Funke, S., Matijevic, D.: Transit: Ultrafast shortest-path queries with linear-time preprocessing. In: 9th DIMACS Implementation Challenge Workshop: Shortest Paths, DIMACS Center, Piscataway, NJ, 13–14 Nov 2006
5. Delling, D., Holzer, M., Muller, K., Schulz, F., Wagner, D.: High-performance multi-level graphs. In: 9th DIMACS Implementation Challenge Workshop: Shortest Paths, DIMACS Center, Piscataway, NJ, 13–14 Nov 2006
6. Delling, D., Sanders, P., Schultes, D., Wagner, D.: Highway hierarchies star. In: 9th DIMACS Implementation Challenge Workshop: Shortest Paths, DIMACS Center, Piscataway, NJ, 13–14 Nov 2006
7. Dijkstra, E.: A note on two problems in connexion with graphs. Numerische Mathematik **1**, 269–271 (1959)
8. Edmonds, N., Breuer, A., Gregor, D., Lumsdaine, A.: Single-source shortest paths with the parallel boost graph library. In: 9th DIMACS Implementation Challenge Workshop: Shortest Paths, DIMACS Center, Piscataway, NJ, 13–14 Nov 2006
9. Goldberg, A., Kaplan, H., Werneck, R.: Better landmarks within reach. In: 9th DIMACS Implementation Challenge Workshop: Shortest Paths. DIMACS Center, Piscataway, NJ, 13–14 Nov 2006
10. Köhler, E., Möhring, R., Schilling, H.: Fast point-to-point shortest path computations with arc-flags. In: 9th DIMACS Implementation Challenge Workshop: Shortest Paths, DIMACS Center, Piscataway, NJ, 13–14 Nov 2006
11. Lauther, U.: An experimental evaluation of point-to-point shortest path calculation on roadnetworks with precalculated edge-flags. In: 9th DIMACS Implementation Challenge Workshop: Shortest Paths, DIMACS Center, Piscataway, NJ, 13–14 Nov 2006
12. Madduri, K., Bader, D., Berry, J., Crobak, J.: Parallel shortest path algorithms for solving large-scale instances. In: 9th DIMACS Implementation Challenge Workshop: Shortest Paths, DIMACS Center, Piscataway, NJ, 13–14 Nov 2006
13. Pascoal, M.: Implementations and empirical comparison of k shortest loopless path algorithms. In: 9th DIMACS Implementation Challenge Workshop: Shortest Paths, DIMACS Center, Piscataway, NJ, 13–14 Nov 2006
14. Sanders, P., Schultes, D.: Robust, almost constant time shortest-path queries in road networks. In: 9th DIMACS Implementation Challenge Workshop: Shortest Paths, DIMACS Center, Piscataway, NJ, 13–14 Nov 2006
15. Santos, J.: K shortest path algorithms. In: 9th DIMACS Implementation Challenge Workshop: Shortest Paths, DIMACS Center, Piscataway, NJ, 13–14 Nov 2006

Implementation Challenge for TSP Heuristics

2002; Johnson, McGeoch

LYLE A. MCGEOCH
Department of Mathematics and Computer Science, Amherst College, Amherst, MA, USA

Keywords and Synonyms

Lin-Kernighan; Two-opt; Three-opt; Held-Karp; TSPLIB; Concorde

Problem Definition

The Eighth DIMACS Implementation Challenge, sponsored by DIMACS, the Center for Discrete Mathematics and Theoretical Computer Science, concerned heuristics for the symmetric Traveling Salesman Problem. The Challenge began in June 2000 and was organized by David S. Johnson, Lyle A. McGeoch, Fred Glover and César Rego. It explored the state-of-the-art in the area of TSP heuristics, with researchers testing a wide range of implementations on a common (and diverse) set of input instances. The Challenge remained ongoing in 2007, with new results still being accepted by the organizers and posted on the Challenge website: www.research.att.com/~dsj/chtsp. A summary of the submissions through 2002 appeared in a book chapter by Johnson and McGeoch [5].

Participants tested their heuristics on four types of instances, chosen to test the robustness and scalability of different approaches:

1. The 34 instances that have at least 1000 cities in TSPLIB, the instance library maintained by Gerd Reinelt.
2. A set of 26 instances consisting of points uniformly distributed in the unit square, with sizes ranging from 1000 to 10,000,000 cities.
3. A set of 23 randomly generated clustered instances, with sizes ranging from 1000 to 316,000 cities.
4. A set of 7 instances based on random distance matrices, with sizes ranging from 1000 to 10,000 cities.

The TSPLIB instances and generators for the random instances are available on the Challenge website. In addition, the website contains a collection of instances for the asymmetric TSP problem.

For each instance upon which a heuristic was tested, the implementers reported the machine used, the tour length produced, the user time, and (if possible) memory usage. Some heuristics could not be applied to all of the instances, either because the heuristics were inherently geometric or because the instances were too large. To help facilitate timing comparisons between heuristics tested on different machines, participants ran a benchmark heuristic (provided by the organizers) on instances of different sizes. The benchmark times could then be used to normalize, at least approximately, the observed running times of the participants' heuristics.

The quality of a tour was computed from a submitted tour length in two ways: as a ratio over the optimal tour length for the instance (if known), and as a ratio over the Held-Karp (HK) lower bound for the instance. The Concorde optimization package of Applegate et al. [1] was able to find the optimum for 58 of the instances in reasonable time. Concorde was used in a second way to compute the HK lower bound for all but the three largest instances. A third algorithm, based on Lagrangian relaxation, was used to compute an approximate HK bound, a lower bound on true HK bound, for the remaining instances. The Challenge website reports on each of these three algorithms, presenting running times and a comparison of the bounds obtained for each instance.

The Challenge website permits a variety of reports to be created:

1. For each heuristic, tables can be generated with results for each instance, including tour length, tour quality, and raw and normalized running times.
2. For each instance, a table can be produced showing the tour quality and normalized running time of each heuristic.
3. For each pair of heuristics, tables and graphs can be produced that compare tour quality and running time for instances of different type and size.

Heuristics for which results were submitted to the Challenge fell into several broad categories:

Heuristics designed for speed. These heuristics – all of which target geometric instances – have running times within a small multiple of the time needed to read the input instance. Examples include the strip and spacefilling-curve heuristics. The speed requirement affects tour quality dramatically. Two of these algorithms produced tours with 14% of the HK lower bound for a particular TSPLIB instance, but none came within 25% on the other 89 instances.

Tour construction heuristics. These heuristics construct tours in various ways, without seeking to find improvements once a single tour passing through all cities is found. Some are simple, such as the nearest-neighbor and greedy heuristics, while others are more complex, such as the famous Christofides heuristic. These heuristics offer a number of options in trading time for tour quality, and several produce tours within 15% of the HK lower bound on most instances in reasonable time. The best of them, a variant of Christofides, produces tours within 8% on uniform instances but is much more time-consuming than the other algorithms.

Simple local improvement heuristics. These include the well-known two-opt and three-opt heuristics and variants of them. These heuristics outperform tour construction heuristics in terms of tour quality on most types of instances. For example, 3-opt gets within about 3% of the HK lower bound on most uniform instances. The submissions in this category explored various implementation choices that affect the time-quality tradeoff.

Lin-Kernighan and its variants. These heuristics extend the local search neighborhood used in 3-opt. Lin-Kernighan can produce high-quality tours (for example, within 2% of the HK lower bound on uniform instances) in reasonable time. One variant, due to Helsgaun [3], obtains tours within 1% on a wide variety of instances, although the running time can be substantial.

Repeated local search heuristics. These heuristics are based on repeated executions of a heuristic such as Lin-Kernighan, with random kicks applied to the tour after a local optimum is found. These algorithms can yield high-quality tours at increased running time.

Heuristics that begin with repeated local search. One example is the tour-merge heuristic [2], which runs repeated local search multiple times, builds a graph containing edges found in the best tours, and does exhaustive search within the resulting graph. This approach yields the best known tours for some of the instances in the Challenge.

The submissions to the Challenge demonstrated the remarkable effectiveness of heuristics for the traveling salesman problem. They also showed that implementation details, such a choice of data structure or whether to approximate aspects of the computation, can affect running time and/or solution quality greatly. Results for a given heuristic also varied enormously depending on the type of instance to which it is applied.

URL to Code

www.research.att.com/~dsj/chtsp

Cross References

► TSP-Based Curve Reconstruction

Recommended Reading

1. Applegate, D., Bixby, R., Chvátal, V., Cook, W.: On the solution of traveling salesman problems. Documenta Mathematica, Extra Volume Proceedings ICM III:645–656. Deutsche Mathematiker-Vereinigung, Berlin (1998)
2. Applegate, D., Bixby, R., Chvátal, V., Cook, W.: Finding tours in the TSP. Technical Report 99885, Research Institute for Discrete Mathematics, Universität Bonn (1999)
3. Helsgaun, K.: An effective implementation of the Lin-Kernighan traveling salesman heuristic. Eur. J. Oper. Res. **126**(1), 106–130 (2000)
4. Johnson, D.S., McGeoch, L.A.: The traveling salesman problem: A case study. In: Aarts, E., Lenstra, J.K. (eds.) Local Search in Combinatorial Optimization, pp. 215–310. Wiley, Chicester (1997)
5. Johnson, D.S., McGeoch, L.A.: Experimental analysis of heuristics for the STSP. In: Gutin, G., Punnen, A.P. (eds.) The Traveling Salesman Problem and Its Variants, pp. 369–443, Kluwer, Dordrecht (2002)

Implementing Shared Registers in Asynchronous Message-Passing Systems

1995; Attiya, Bar-Noy, Dolev

ERIC RUPPERT
Department Computer Science and Engineering, York University, Toronto, ON, Canada

Keywords and Synonyms

Simulation; Emulation

Problem Definition

A distributed system is composed of a collection of n processes which communicate with one another. Two means of interprocess communication have been heavily studied. *Message-passing systems* model computer networks where each process can send information over message channels to other processes. In *shared-memory systems*, processes communicate less directly by accessing information in shared data structures. Distributed algorithms are often easier to design for shared-memory systems because of their similarity to single-process system architectures. However, many real distributed systems are constructed as message-passing systems. Thus, a key problem in distributed computing is the implementation of shared memory in message-passing systems. Such implementations are also called simulations or emulations of shared memory.

The most fundamental type of shared data structure to implement is a *(read-write) register*, which stores a value, taken from some domain D. It is initially assigned a value from D and can be accessed by two kinds of operations, read and write(v), where $v \in D$. A register may be either *single-writer*, meaning only one process is allowed to write it, or *multi-writer*, meaning any process may write to it. Similarly, it may be either *single-reader* or *multi-reader*. Attiya and Welch [4] give a survey of how to build multi-writer, multi-reader registers from single-writer, single-reader ones.

If reads and writes are performed one at a time, they have the following effects: a read returns the value stored in the register to the invoking process, and a write(v) changes the value stored in the register to v and returns an acknowledgment, indicating that the operation is complete. When many processes apply operations concurrently, there are several ways to specify a register's behavior [14]. A single-writer register is *regular* if each read returns either the argument of the write that completed most recently before the read began or the argument of some write operation that runs concurrently with the read. (If there is no write that completes before the read begins, the read may return either the initial value of the register or the value of a concurrent write operation.) A register is *atomic* (see ▶ linearizability) if each operation appears to take place instantaneously. More precisely, for any concurrent execution, there is a total order of the operations such that each read returns the value written by the last write that precedes it in the order (or the initial value of the register, if there is no such write). Moreover, this total order must be consistent with the temporal order of operations: if one operation finishes before another one begins, the former must precede the latter in the total order. Atomicity is a stronger condition than regularity, but it is possible to implement atomic registers from regular ones with some complexity overhead [12].

This article describes the problem of implementing registers in an asynchronous message-passing system in which processes may experience crash failures. Each process can send a message, containing a finite string, to any other process. To make the descriptions of algorithms more uniform, it is often assumed that processes can send messages to themselves. All messages are eventually delivered. In the algorithms described below, senders wait for an acknowledgment of each message before sending the next message, so it is not necessary to assume that the message channels are first-in-first-out. The system is totally asynchronous: there is no bound on the time required for a message to be delivered to its recipient or for a process to perform a step of local computation. A process that fails by crashing stops executing its code, but other processes cannot distinguish between a process that has crashed and one that is running very slowly. (Failures of message channels [3] and more malicious kinds of process failures [15] have also been studied.)

A *t-resilient* register implementation provides programmes to be executed by processes to simulate read and write operations. These programmes can include any standard control structures and accesses to a process's local memory, as well as instructions to send a message to another process and to read the process's buffer, where incoming messages are stored. The implementation should also specify how the processes' local variables are initialized to reflect any initial value of the implemented register. In the case of a single-writer register, only one process may execute the write programme. A process may invoke the read and write programmes repeatedly, but it must wait for one invocation to complete before starting the next one. In any such execution where at most t processes crash, each of a process's invocations of the read or write programme should eventually terminate. Each read operation returns a result from the set D, and these results should satisfy regularity or atomicity.

Relevant measures of algorithm complexity include the number of messages transmitted in the system to perform an operation, the number of bits per message, and the amount of local memory required at each process. One measure of time complexity is the time needed to perform an operation, under the optimistic assumption that the time to deliver messages is bounded by Δ and local computation is instantaneous (although algorithms must work correctly even without these assumptions).

Key Results

Implementing a Regular Register

One of the core ideas for implementing shared registers in message-passing systems is a construction that implements a regular single-writer multi-reader register. It was introduced by Attiya, Bar-Noy and Dolev [3] and made more explicit by Attiya [2]. A write(v) sends the value v to all processes and waits until a majority of the processes ($\lfloor \frac{n}{2} \rfloor + 1$, including the writer itself) return an acknowledgment. A reader sends a request to all processes for their latest values. When it has received responses from a majority of processes, it picks the most recently written value among them. If a write completes before a read begins, at least one process that answers the reader has received the write's value prior to sending its response to the reader. This is because any two sets that each contain a majority of the processes must overlap. The time required by operations when delivery times are bounded is 2Δ.

This algorithm requires the reader to determine which of the values it receives is most recent. It does this using *timestamps* attached to the values. If the writer uses increasing integers as timestamps, the messages grow without bound as the algorithm runs. Using the bounded timestamp scheme of Israeli and Li [13] instead yields the following theorem.

Theorem 1 (Attiya [2]) *There is an $\lceil \frac{n-2}{2} \rceil$-resilient implementation of a regular single-writer, multi-reader register in a message-passing system of n processes. The implementation uses $\Theta(n)$ messages per operation, with $\Theta(n^3)$ bits per message. The writer uses $\Theta(n^4)$ bits of local memory and each reader uses $\Theta(n^3)$ bits.*

Theorem 1 is optimal in terms of fault-tolerance. If $\lceil \frac{n}{2} \rceil$ processes can crash, the network can be partitioned into two halves of size $\lfloor \frac{n}{2} \rfloor$, with messages between the two halves delayed indefinitely. A write must terminate before any evidence of the write is propagated to the half not containing the writer, and then a read performed by a process in that half cannot return an up-to-date value. For $t \geq \lceil \frac{n}{2} \rceil$, registers can be implemented in a message-passing system only if some degree of synchrony is present in the system. The exact amount of synchrony required was studied by Delporte-Gallet et al. [6].

Theorem 1 is within a constant factor of the optimal number of messages per operation. Evidence of each write must be transmitted to at least $\lceil \frac{n}{2} \rceil - 1$ processes, requiring $\Omega(n)$ messages; otherwise this evidence could be obliterated by crashes. A write must terminate even if only $\lfloor \frac{n}{2} \rfloor + 1$ processes (including the writer) have received information about the value written, since the rest of the processes could have crashed. Thus, a read must receive information from at least $\lceil \frac{n}{2} \rceil$ processes (including itself) to ensure that it is aware of the most recent write operation.

A t-resilient implementation, for $t < \lceil \frac{n}{2} \rceil$, that uses $\Theta(t)$ messages per operation is obtained by the following adaptation. A set of $2t + 1$ processes is preselected to be data storage servers. Writes send information to the servers, and wait for $t + 1$ acknowledgments. Reads wait for responses from $t + 1$ of the servers and choose the one with the latest timestamp.

Implementing an Atomic Register

Attiya, Bar-Noy and Dolev [3] gave a construction of an atomic register in which readers forward the value they return to all processes and wait for an acknowledgment from a majority. This is done to ensure that a read does not return an older value than another read that precedes it. Using unbounded integer timestamps, this algorithm uses $\Theta(n)$ messages per operation. The time needed per operation when delivery times are bounded is 2Δ for writes and 4Δ for reads. However, their technique of bounding the timestamps increases the number of messages per operation to $\Theta(n^2)$ (and the time per operation to 12Δ). A better implementation of atomic registers with bounded message size is given by Attiya [2]. It uses the regular registers of Theorem 1 to implement atomic registers using the "handshaking" construction of Haldar and Vidyasankar [12], yielding the following result.

Theorem 2 (Attiya [2]) *There is an $\lceil \frac{n-2}{2} \rceil$-resilient implementation of an atomic single-writer, multi-reader register in a message-passing system of n processes. The implementation uses $\Theta(n)$ messages per operation, with $\Theta(n^3)$ bits per message. The writer uses $\Theta(n^5)$ bits of local memory and each reader uses $\Theta(n^4)$ bits.*

Since atomic registers are regular, this algorithm is optimal in terms of fault-tolerance and within a constant factor of optimal in terms of the number of messages. The time used when delivery times are bounded is at most 14Δ for writes and 18Δ for reads.

Applications

Any distributed algorithm that uses shared registers can be adapted to run in a message-passing system using the implementations described above. This approach yielded new or improved message-passing solutions for a number of problems, including randomized consensus [1], multi-writer registers [4], and snapshot objects ► Snapshots. The reverse simulation is also possible, using a straightforward implementation of message channels by single-writer, single-reader registers. Thus, the two asynchronous models are equivalent, in terms of the set of problems that they can solve, assuming only a minority of processes crash. However there is some complexity overhead in using the simulations.

If a shared-memory algorithm is implemented in a message-passing system using the algorithms described here, processes must continue to operate even when the algorithm terminates, to help other processes execute their reads and writes. This cannot be avoided: if each process must stop taking steps when its algorithm terminates, there are some problems solvable with shared registers that are not solvable in the message-passing model [5].

Using a majority of processes to "validate" each read and write operation is an example of a quorum system, originally introduced for replicated data by Gifford [10]. In general, a quorum system is a collection of sets of processes, called quorums, such that every two quorums intersect. Quorum systems can also be designed to implement shared registers in other models of message-passing systems, including dynamic networks and systems with malicious failures. For examples, see [7,9,11,15].

Open Problems

Although the algorithms described here are optimal in terms of fault-tolerance and message complexity, it is not known if the number of bits used in messages and local memory is optimal. The exact time needed to do reads and writes when messages are delivered within time Δ is also a topic of ongoing research. (See, for example, [8].) As mentioned above, the simulation of shared registers can be used to implement shared-memory algorithms in message-passing systems. However, because the simulation introduces considerable overhead, it is possible that some of those problems could be solved more efficiently by algorithms designed specifically for message-passing systems.

Cross References

▶ Linearizability
▶ Quorums
▶ Registers

Recommended Reading

1. Aspnes, J.: Randomized protocols for asynchronous consensus. Distrib. Comput. **16**(2–3), 165–175 (2003)
2. Attiya, H.: Efficient and robust sharing of memory in message-passing systems. J. Algorithms **34**(1), 109–127 (2000)
3. Attiya, H., Bar-Noy, A., Dolev, D.: Sharing memory robustly in message-passing systems. J. ACM **42**(1), 124–142 (1995)
4. Attiya, H., Welch, J.: Distributed Computing: Fundamentals, Simulations and Advanced Topics, 2nd edn. Wiley-Interscience, Hoboken (2004)
5. Chor, B., Moscovici, L.: Solvability in asynchronous environments. In: Proc. 30th Symposium on Foundations of Computer Science, pp. 422–427 (1989)
6. Delporte-Gallet, C., Fauconnier, H., Guerraoui, R., Hadzilacos, V., Kouznetsov, P., Toueg, S.: The weakest failure detectors to solve certain fundamental problems in distributed computing. In: Proc. 23rd ACM Symposium on Principles of Distributed Computing, pp. 338–346. St. John's, Newfoundland, 25–28 July 2004
7. Dolev, S., Gilbert, S., Lynch, N.A., Shvartsman, A.A., Welch, J.L.: GeoQuorums: Implementing atomic memory in mobile ad hoc networks. Distrib. Comput. **18**(2), 125–155 (2005)
8. Dutta, P., Guerraoui, R., Levy, R.R., Chakraborty, A.: How fast can a distributed atomic read be? In: Proc. 23rd ACM Symposium on Principles of Distributed Computing, pp. 236–245. St. John's, Newfoundland, 25–28 July 2004
9. Englert, B., Shvartsman, A.A.: Graceful quorum reconfiguration in a robust emulation of shared memory. In: Proc. 20th IEEE International Conference on Distributed Computing Systems, pp. 454–463. Taipei, 10–13 April 2000
10. Gifford, D.K.: Weighted voting for replicated data. In: Proc. 7th ACM Symposium on Operating Systems Principles, pp. 150–162. Pacific Grove, 10–12 December 1979
11. Gilbert, S., Lynch, N., Shvartsman, A.: Rambo II: rapidly reconfigurable atomic memory for dynamic networks. In: Proc. International Conference on Dependable Systems and Networks, pp. 259–268. San Francisco, 22–25 June 2003
12. Haldar, S., Vidyasankar, K.: Constructing 1-writer multireader multivalued atomic variables from regular variables. J. ACM **42**(1), 186–203 (1995)
13. Israeli, A., Li, M.: Bounded time-stamps. Distrib. Comput. **6**(4), 205–209 (1993)
14. Lamport, L.: On interprocess communication, Part II: Algorithms. Distrib. Comput. **1**(2), 86–101 (1986)
15. Malkhi, D., Reiter, M.: Byzantine quorum systems. Distrib. Comput. **11**(4), 203–213 (1998)

Incentive Compatible Algorithms

▶ Computing Pure Equilibria in the Game of Parallel Links
▶ Truthful Mechanisms for One-Parameter Agents

Incentive Compatible Selection

2006; Chen, Deng, Liu

XI CHEN[1], XIAOTIE DENG[2]
[1] Department of Computer Science and Technology, Tsinghua University, Beijing, China
[2] Department of Computer Science, City University of Hong Kong, Hong Kong, China

Keywords and Synonyms

Incentive compatible selection; Incentive compatible ranking; Algorithmic mechanism design

Problem Definition

Ensuring truthful evaluation of alternatives in human activities has always been an important issue throughout history. In sport, in particular, such an issue is vital and practice of the fair play principle has been consistently put forward as a matter of foremost priority. In addition to relying on the code of ethics and professional responsibility of players and coaches, the design of game rules is an important measure in enforcing fair play.

Ranking alternatives through pairwise comparisons (or competitions) is the most common approach in sports tournaments. Its goal is to find out the "true" ordering among alternatives through complete or partial pairwise competitions [1, 3, 4, 5, 6, 7]. Such studies have been mainly based on the assumption that all the players play truthfully, i. e., with their maximal effort. It is, however, possible that some players form a coalition, and cheat for group benefit. An interesting example can be found in [2].

Problem Description

The work of Chen, Deng, and Liu [2] considers the problem of choosing m winners out of n candidates.

Suppose a tournament is held among n players $P_n = \{p_1, \dots p_n\}$ and m winners are expected to be selected by a selection protocol. Here a protocol $f_{n,m}$ is a predefined function (which will become clear later) to choose winners through pairwise competitions, with the intention of finding m players of highest capacity. When the tournament starts, a distinct ID in $N_n = \{1, 2, \dots n\}$ is assigned to each player in P_n by a randomly picked indexing function $I : P_n \to N_n$. Then a match is played between each pair of players. The competition outcomes will form a graph G, whose vertex set is N_n and edges represent the results of all the matches. Finally, the graph will be treated as the input to $f_{n,m}$, and it will output a set of m winners. Now it should be clear that $f_{n,m}$ maps every possible tournament graph G to a subset (of cardinality m) of N_n.

Suppose there exists a group of bad players who play dishonestly, i. e. they might lose a match on purpose to gain overall benefit for the whole group, while the rest of the players always play truthfully, i. e. they try their best to win matches. The group of bad players gains benefit if they are able to have more winning positions than that according to the true ranking. Given knowledge of the selection protocol $f_{n,m}$, the indexing function I and the true ranking of all players, the bad players try to find a cheating strategy that can fool the protocol and gain benefit.

The problem is discussed under two models in which the characterizations of bad players are different. Under the *collective incentive compatible model*, bad players are willing to sacrifice themselves to win group benefit; while the ones under the *alliance incentive compatible model* only cooperate if their individual interests are well maintained in the cheating strategy.

The goal is to find an "ideal" protocol, under which players or groups of players maximize their benefits only by strictly following the fair play principle, i. e. always play with maximal effort.

Formal Definitions

When the tournament begins, an indexing function I is randomly picked, which assigns ID $I(p) \in N_n$ to each player $p \in P_n$. Then a match is played between each pair of players, and the results are represented as a directed graph G. Finally, G is fed into the predefined selection protocol $f_{n,m}$, to produce a set of m winners $I^{-1}(W)$, where $W = f_{n,m}(G) \subset N_n$.

Notations An indexing function I for a tournament attended by n players $P_n = \{p_1, p_2, \dots p_n\}$ is a one-to-one correspondence from P_n to the set of *IDs*: $N_n = \{1, 2, \dots n\}$. A ranking function R is a one-to-one correspondence from P_n to $\{1, 2, \dots n\}$. $R(p)$ represents the underlying true ranking of player p among the n players. The smaller, the stronger.

A tournament graph of size n is a directed graph $G = (N_n, E)$ such that, for all $i \neq j \in N_n$, either $ij \in E$ (player with ID i beats player with ID j) or $ji \in E_n$. Let K_n denote the set of all such graphs. A selection protocol $f_{n,m}$, which chooses m winners out of n candidates, is a function from K_n to $\{S \subset N_n \text{ and } |S| = m\}$.

A tournament T_n among players P_n is a pair $T_n = (R, B)$ where R is a ranking function from P_n to N_n and $B \subset P_n$ is the group of bad players.

Definition 1 (Benefit) Given a protocol $f_{n,m}$, a tournament $T_n = (R, B)$, an indexing function I and a tournament graph $G \in K_n$, the benefit of the group of bad players is

$$\mathbf{Ben}(f_{n,m}, T_n, I, G) = \left|\left\{i \in f_{n,m}(G),\ I^{-1}(i) \in B\right\}\right| - \left|\left\{p \in B,\ R(p) \le m\right\}\right|.$$

Given knowledge of $f_{n,m}$, T_n and I, not every $G \in K_n$ is a feasible strategy for B: the group of bad players. First, it depends on the tournament $T_n = (R, B)$, e. g. a player $p_b \in B$ cannot win a player $p_g \notin B$ if $R(p_b) > R(p_g)$. Second, it depends on the property of bad players which is specified by the model considered. Tournament graphs, which are recognized as feasible strategies, are characterized below, for each model. The key difference is that, a bad player in the alliance incentive compatible model is not willing to sacrifice his own winning position, while a player in the other model fights for group benefit at all costs.

Definition 2 (Feasible Strategy) Given $f_{n,m}$, $T_n = (R, B)$ and I, graph $G \in K_n$ is *c-feasible* if

1 For every two players $p_i, p_j \notin B$, if $R(p_i) < R(p_j)$, then $I(p_i)I(p_j) \in E$;
2 For all $p_g \notin B$ and $p_b \in B$, if $R(p_g) < R(p_b)$, then edge $I(p_g)I(p_b) \in E$.

Graph $G \in K_n$ is *a-feasible* if it is c-feasible and also satisfies

3 For every bad player $p \in B$, if $R(p) \le m$, then $I(p) \in f_{n,m}(G)$.

A cheating strategy is then a feasible tournament graph G that can be employed by the group of bad players to gain positive benefit.

Definition 3 (Cheating Strategy) Given $f_{n,m}$, $T_n = (R, B)$ and I, a cheating strategy for the group of bad players under the collective incentive compatible (alliance incentive compatible) model is a graph $G \in K_n$ which is c-feasible (a-feasible) and satisfies $\boldsymbol{Ben}(f_{n,m}, T_n, I, G) > 0$.

The following two problems are studied in [2]: (**1**) Is there a protocol $f_{n,m}$ such that for all T_n and I, no cheating strategy exists under the collective incentive compatible model? (**2**) Is there a protocol $f_{n,m}$ such that for all T_n and I, no cheating strategy exists under the alliance incentive compatible model?

Key Results

Definition 4 For all integers n and m such that $2 \le m \le n - 2$, a tournament graph $G_{n,m} = (N_n, E) \in K_n$, which consists of three parts T_1, T_2, and T_3, is defined as follows:

1 $T_1 = \{1, 2, \dots m-2\}$. For all $i < j \in T_1$, edge $ij \in E$;
2 $T_2 = \{m-1, m, m+1\}$. $(m-1)m$, $m(m+1)$, $(m+1)(m-1) \in E$;
3 $T_3 = \{m+2, m+3, \dots n\}$. For all $i < j \in T_3$, edge $ij \in E$;
4 For all $i' \in T_i$ and $j' \in T_j$ such that $i < j$, edge $i'j' \in E$.

Theorem 1 *Under the collective incentive compatible model, for every selection protocol $f_{n,m}$ with $2 \le m \le n-2$, if $T_n = (R, B)$ satisfies:* **(1)** *At least one bad player ranks as high as $m-1$;* **(2)** *The ones ranked $m+1$ and $m+2$ are both bad players;* **(3)** *The one ranked m is a good player, then there always exists an indexing function I such that $G_{n,m}$ is a cheating strategy.*

Theorem 2 *Under the alliance incentive compatible model, if $n - m \ge 3$, then there exists a selection protocol $f_{n,m}$* [2] *such that, for every tournament T_n, indexing function I and a-feasible strategy $G \in K_n$,* $\mathbf{Ben}(f_{n,m}, T_n, I, G) \le 0$.

Applications

The result shows that, if players are willing to sacrifice themselves, no protocol is able to prevent malicious coalitions from obtaining undeserved benefits.

The result may have potential applications in the design of output truthful mechanisms.

Open Problems

Under the collective incentive compatible model, the work of Chen, Deng, and Liu indicates that cheating strategies are available in at least 1/8 tournaments, assuming the probability for each player to be in the bad group is 1/2. Could this bound be improved? Or could one find a good selection protocol in the sense that the number of tournaments with cheating strategies is close to this bound? On the other hand, although no ideal protocol exists in this model, does there exist any randomized protocol, under which the probability of having cheating strategies is negligible?

Cross References

► Algorithmic Mechanism Design
► Truthful Multicast

Recommended Reading

1. Chang, P., Mendonca, D., Yao, X., Raghavachari, M.: An evaluation of ranking methods for multiple incomplete round-robin tournaments. In: Proceedings of the 35th Annual Meeting of Decision Sciences Institute, Boston, 20–23 November 2004
2. Chen, X., Deng, X., Liu, B.J.: On incentive compatible competitive selection protocol. In: COCOON'06: Proceedings of the 12th Annual International Computing and Combinatorics Conference, pp. 13–22, Taipei, 15–18 August 2006
3. Harary, F., Moser, L.: The theory of round robin tournaments. Am. Math. Mon. **73**(3), 231–246 (1966)
4. Jech, T.: The ranking of incomplete tournaments: A mathematician's guide to popular sports. Am. Math. Mon. **90**(4), 246–266 (1983)
5. Mendonca, D., Raghavachari, M.: Comparing the efficacy of ranking methods for multiple round-robin tournaments. Eur. J. Oper. Res. **123**, 593–605 (1999)
6. Rubinstein, A.: Ranking the participants in a tournament. SIAM J. Appl. Math. **38**(1), 108–111 (1980)
7. Steinhaus, H.: Mathematical Snapshots. Oxford University Press, New York (1950)

Incremental Algorithms

► Fully Dynamic Connectivity
► Fully Dynamic Transitive Closure

Independent Sets in Random Intersection Graphs

2004; Nikoletseas, Raptopoulos, Spirakis

SOTIRIS NIKOLETSEAS, CHRISTOFOROS RAPTOPOULOS, PAUL SPIRAKIS
Research Academic Computer Technology Institute, Greece and Computer Engineering and Informatics Department, University of Patras, Patras, Greece

Keywords and Synonyms

Existence and efficient construction of independent sets of vertices in general random intersection graphs

Problem Definition

This problem is concerned with the efficient construction of an independent set of vertices (i. e. a set of vertices with no edges between them) with maximum cardinality, when the input is an instance of the uniform random intersection graphs model. This model was introduced by Karoński, Sheinerman, and Singer-Cohen in [4] and Singer-Cohen in [10] and it is defined as follows

Definition 1 (Uniform random intersection graph) Consider a universe $M = \{1, 2, \dots, m\}$ of elements and a set of vertices $V = \{v_1, v_2, \dots, v_n\}$. If one assigns independently to each vertex v_j, $j = 1, 2, \dots, n$, a subset S_{v_j} of M by choosing each element independently with probability p and puts an edge between two vertices v_{j_1}, v_{j_2} if and only if $S_{v_{j_1}} \cap S_{v_{j_2}} \neq \emptyset$, then the resulting graph is an instance of the uniform random intersection graph $G_{n,m,p}$.

The universe M is sometimes called *label set* and its elements *labels*. Also, denote by L_l, for $l \in M$, the set of vertices that have chosen label l.

Because of the dependence of edges, this model can abstract more accurately (than the Bernoulli random graphs model $G_{n,p}$ that assumes independence of edges) many real-life applications. Furthermore, Fill, Sheinerman, and Singer-Cohen show in [3] that for some ranges of the parameters n, m, p ($m = n^{\alpha}, \alpha > 6$), the spaces $G_{n,m,p}$ and $G_{n,\hat{p}}$ are equivalent in the sense that the total variation distance between the graph random variables has limit 0. The work of Nikoletseas, Raptopoulos, and Spirakis [7] introduces two new models, namely the *general random intersection graphs model* $G_{n,m,\vec{p}}$, $\vec{p} = [p_1, p_2, \ldots, p_m]$ and the *regular random intersection graphs model* $G_{n,m,\lambda}$, $\lambda > 0$ that use a different way to randomly assign labels to vertices, but the edge appearance rule remains the same. The $G_{n,m,\vec{p}}$ model is a generalization of the uniform model where each label $i \in M$ is chosen independently with probability p_i, whereas in the $G_{n,m,\lambda}$ model each vertex chooses a random subset of M with exactly λ labels.

The authors in [7] first consider the existence of independent sets of vertices of a given cardinality in general random intersection graphs and provide exact formulae for the mean and variance of the number of independent sets of vertices of cardinality k. Furthermore, they present and analyze three polynomial time (on the number of labels m and the number of vertices n) algorithms for constructing large independent sets of vertices when the input is an instance of the $G_{n,m,p}$ model. To the best knowledge of the entry authors, this work is the first to consider algorithmic issues for these models of random graphs.

Key Results

The following theorems concern the existence of independent sets of vertices of cardinality k in general random intersection graphs. The proof of Theorem 1 uses the linearity of expectation of sums of random variables.

Theorem 1 *Let $X^{(k)}$ denote the number of independent sets of size k in a random intersection graph $G(n, m, \vec{p})$, where $\vec{p} = [p_1, p_2, \ldots, p_m]$. Then*

$$E\left[X^{(k)}\right] = \binom{n}{k} \prod_{i=1}^{m} \left((1-p_i)^k + kp_i(1-p_i)^{k-1}\right) .$$

Theorem 2 *Let $X^{(k)}$ denote the number of independent sets of size k in a random intersection graph $G(n, m, \vec{p})$, where $\vec{p} = [p_1, p_2, \ldots, p_m]$. Then*

$$\operatorname{Var}\left(X^{(k)}\right) = \sum_{s=1}^{k} \binom{n}{2k-s}\binom{2k-s}{s} \left(\gamma(k,s)\frac{E\left[X^{(k)}\right]}{\binom{n}{k}} - \frac{E^2\left[X^{(k)}\right]}{\binom{n}{k}^2}\right)$$

where $E\left[X^{(k)}\right]$ is the mean number of independent sets of size k and

$$\gamma(k,s) = \prod_{i=1}^{m}\left((1-p_i)^{k-s} + (k-s)p_i(1-p_i)^{k-s-1} \left(1 - \frac{sp_i}{1+(k-1)p_i}\right)\right) .$$

Theorem 2 is proved by first writing the variance as the sum of covariances and then applying a vertex contraction technique that merges several vertices into one supervertex with similar probabilistic behavior in order to compute the covariances. By using the second moment method (see [1]) one can derive thresholds for the existence of independent sets of size k.

One of the three algorithms that were proposed in [7] is presented below. The algorithm starts with V (i. e. the set of vertices of the graph) as its "candidate" independent set. In every subsequent step it chooses a label and removes from the current candidate independent set all vertices having that label in their assigned label set except for one. Because of the edge appearance rule, this ensures that after doing this for every label in M, the final candidate independent set will contain only vertices that do not have edges between them and so it will be indeed an independent set.

Algorithm:
Input: A random intersection graph $G_{n,m,p}$.
Output: An independent set of vertices A_m.

```
1. set $A_0 := V$; set $L := M$;
2. for $i = 1$ to $m$ do
3. begin
4.     select a random label $l_i \in L$; set $L := L - \{l_i\}$;
5.     set $D_i := \{v \in A_{i-1} : l_i \in S_v\}$;
6.     if ($|D_i| \geq 1$) then select a random vertex $u \in D_i$ and
           set $D_i := D_i - \{u\}$;
7.     set $A_i := A_{i-1} - D_i$;
8. end
9. output $A_m$;
```

The following theorem concerns the cardinality of the independent set produced by the algorithm. The analysis of the algorithm uses Wald's equation (see [9]) for sums of a random number of random variables to calculate the mean value of $|A_m|$, and also Chernoff bounds (see e. g. [6]) for concentration around the mean.

Theorem 3 *For the case $mp = \alpha \log n$, for some constant $\alpha > 1$ and $m \geq n$, and for some constant $\beta > 0$, the following hold with high probability:*

1. *If $np \to \infty$ then $|A_m| \geq (1-\beta)\frac{n}{\log n}$.*
2. *If $np \to b$ where $b > 0$ is a constant then $|A_m| \geq (1-\beta)n(1-\mathrm{e}^{-b})$.*
3. *If $np \to 0$ then $|A_m| \geq (1-\beta)n$.*

The above theorem shows that the algorithm manages to construct a quite large independent set with high probability.

Applications

First of all, note that (as proved in [5]) any graph can be transformed into an intersection graph. Thus, the random intersection graphs models can be very general. Furthermore, for some ranges of the parameters n, m, p ($m = n^{\alpha}, \alpha > 6$) the spaces $G_{n,m,p}$ and $G_{n,p}$ are equivalent (as proved by Fill, Sheinerman, and Singer-Cohen in [3], showing that in this range the total variation distance between the graph random variables has limit 0).

Second, random intersection graphs (and in particular the general intersection graphs model of [7]) may model real-life applications more accurately (compared to the $G_{n,p}$ case). In particular, such graphs can model resource allocation in networks, e. g. when network nodes (abstracted by vertices) access shared resources (abstracted by labels): the intersection graph is in fact the conflict graph of such resource allocation problems.

Other Related Work

In their work [4] Karoński et al. consider the problem of the emergence of graphs with a constant number of vertices as induced subgraphs of $G_{n,m,p}$ graphs. By observing that the $G_{n,m,p}$ model generates graphs via clique covers (for example the sets $L_l, l \in M$ constitute an obvious clique cover) they devise a natural way to use them together with the first and second moment methods in order to find thresholds for the appearance of any fixed graph H as an induced subgraph of $G_{n,m,p}$ for various values of the parameters n, m and p.

The connectivity threshold for $G_{n,m,p}$ was considered by Singer-Cohen in [10]. She studies the case $m = n^{\alpha}, \alpha > 0$ and distinguishes two cases according to the value of α. For the case $\alpha > 1$, the results look similar to the $G_{n,p}$ graphs, as the mean number of edges at the connectivity thresholds are (roughly) the same. On the other hand, for $\alpha \leq 1$ we get denser graphs in the $G_{n,m,p}$ model. Besides connectivity, [10] examines also the size of the largest clique in uniform random intersection graphs for certain values of n, m and p.

The existence of Hamilton cycles in $G_{n,m,p}$ graphs was considered by Efthymiou and Spirakis in [2]. The authors use coupling arguments to show that the threshold of appearance of Hamilton cycles is quite close to the connectivity threshold of $G_{n,m,p}$. Efficient probabilistic algorithms for finding Hamilton cycles in uniform random intersection graphs were presented by Raptopoulos and Spirakis in [8]. The analysis of those algorithms verify that they perform well w.h.p. even for values of p that are close to the connectivity threshold of $G_{n,m,p}$. Furthermore, in the same work, an expected polynomial algorithm for finding Hamilton cycles in $G_{n,m,p}$ graphs with constant p is given.

In [11] Stark gives approximations of the distribution of the degree of a fixed vertex in the $G_{n,m,p}$ model. More specifically, by applying a sieve method, the author provides an exact formula for the probability generating function of the degree of some fixed vertex and then analyzes this formula for different values of the parameters n, m and p.

Open Problems

A number of problems related to random intersection graphs remain open. Nearly all the algorithms proposed so far concerning constructing large independent sets and finding Hamilton cycles in random intersection graphs are greedy. An interesting and important line of research would be to find more sophisticated algorithms for these problems that outperform the greedy ones. Also, all these algorithms were presented and analyzed in the uniform random intersection graphs model. Very little is known about how the same algorithms would perform when their input was an instance of the general or even the regular random intersection graph models.

Of course, many classical problems concerning random graphs have not yet been studied. One such example is the size of the minimum dominating set (i. e. a set of vertices that has the property that all vertices of the graph either belong to this set or are connected to it) in a random intersection graph. Also, what is the degree sequence of $G_{n,m,p}$ graphs? Note that this is very different from the problem addressed in [11].

Finally, notice that none of the results presented in the bibliography for general or uniform random intersection graphs carries over immediately to regular random intersection graphs. Of course, for some values of n, m, p and

λ, certain graph properties shown for $G_{n,m,p}$ could also be proved for $G_{n,m,\lambda}$ by showing concentration of the number of labels chosen by any vertex via Chernoff bounds. Other than that, the fixed sizes of the sets assigned to each vertex impose more dependencies to the model.

Cross References

► Hamilton Cycles in Random Intersection Graphs

Recommended Reading

1. Alon, N., Spencer, H.: The Probabilistic Method. Wiley, Inc. (2000)
2. Efthymiou, C., Spirakis, P.: On the existence of hamiltonian cycles in random intersec- tion graphs. In: Proceedings of 32st International colloquium on Automata, Languages and Programming (ICALP), pp. 690–701. Springer, Berlin Heidelberg (2005)
3. Fill, J.A., Sheinerman, E.R., Singer-Cohen, K.B.: Random intersection graphs when $m = \omega(n)$: An equivalence theorem relating the evolution of the g(n, m, p) and g(n, p) models. Random Struct. Algorithm. **16**(2), 156–176 (2000)
4. Karoński, M., Scheinerman, E.R., Singer-Cohen, K.B.: On random intersection graphs: The subgraph problem. Adv. Appl. Math. **8**, 131–159 (1999)
5. Marczewski, E.: Sur deux propriétés des classes d' ensembles. Fund. Math. **33**, 303–307 (1945)
6. Motwani, R., Raghavan, P.: Randomized Algorithms. Cambridge University Press (1995)
7. Nikoletseas, S., Raptopoulos, C., Spirakis, P.: The existence and efficient construction of large independent sets in general random intersection graphs. In: Proceedings of 31st International colloquium on Automata, Languages and Programming (ICALP), pp. 1029–1040. Springer, Berlin Heidelberg (2004) Also in the Theoretical Computer Science (TCS) Journal, accepted, to appear in 2008
8. Raptopoulos, C., Spirakis, P.: Simple and efficient greedy algorithms for hamiltonian cycles in random intersection graphs. In: Proceedings of the 16th International Symposium on Algorithms and Computation (ISAAC), pp 493–504. Springer, Berlin Heidelberg (2005)
9. Ross, S.: Stochastic Processes. Wiley (1995)
10. Singer-Cohen, K.B.: Random Intersection Graphs. Ph. D. thesis, John Hopkins University, Balimore (1995)
11. Stark, D.: The vertex degree distribution of random intersection graphs. Random Struct. Algorithms **24**, 249–258 (2004)

Indexed Approximate String Matching

2006; Chan, Lam, Sung, Tam, Wong

WING-KIN SUNG
Department of Computer Science, National University of Singapore, Singapore, Singapore

Keywords and Synonyms

Indexed inexact pattern matching problem; Indexed pattern searching problem based on hamming distance or edit distance; Indexed k-mismatch problem; Indexed k-difference problem

Problem Definition

Consider a text $S[1..n]$ over a finite alphabet Σ. One wants to build an index for S such that for any query pattern $P[1..m]$ and any integer $k \geq 0$, one can report efficiently all locations in S that match P with at most k errors. If error is measured in terms of the Hamming distance (number of character substitutions), the problem is called the k-mismatch problem. If error is measured in term of the edit distance (number of character substitutions, insertions or deletions), the problem is called the k-difference problem. The two problems are formally defined as follows.

Problem 1 (k-mismatch problem) *Consider a text $S[1..n]$ over a finite alphabet Σ. For any pattern P and threshold k, position i is an occurrence of P if the hamming distance between P and $S[i..i']$ is less than k for some i'. The k-mismatch problem asks for an index I for S such that, for any pattern P, one can report all occurrences of P in S efficiently.*

Problem 2 (k-difference problem) *Consider a text $S[1..n]$ over a finite alphabet Σ. For any pattern P and threshold k, position i is an occurrence of P if the edit distance between P and $S[i..i']$ is less than k for some i'. The k-difference problem asks for an index I for S such that, for any pattern P, one can report all occurrences of P in S efficiently.*

The major concern of the two problems is how to achieve efficient pattern searching without using a large amount of space for storing the index.

Below, assume $|\Sigma|$ (the size of the alphabet) is constant.

Key Results

Table 1 summarizes the related results in the literature. Below, briefly describes the current best results.

For indexes for exact matching ($k = 0$), the best results utilize data structures like the suffix tree, compressed suffix array, and FM-index. Theorems 1 and 2 describe those results.

Theorem 1 (Weiner, 1973 [17]) *Given a suffix tree of size $O(n)$ words, one can support exact (0-mismatch) matching in $O(m + occ)$ time where occ is the number of occurrences.*

Indexed Approximate String Matching, Table 1

Known results for k-difference matching. c is some positive constant and ε is some positive constant smaller than 1

Space	$k = 1$	
$O(n \log^2 n)$ words	$O(m \log n \log\log n + occ)$	[1]
$O(n \log n)$ words	$O(m \log\log n + occ)$	[2]
	$O(m + occ + \log n \log\log n)$	[8]
$O(n)$ words	$O(\min\{n, m^2\} + occ)$	[6]
	$O(m \log n + occ)$	[11]
	$O(kn^{\epsilon} \log n)$	[14]
	$O(n^{\epsilon})$	[15]
	$O(m + occ + \log^3 n \log\log n)$	[3]
	$O(m + occ + \log n \log\log n)$	[4]
$O(n\sqrt{\log n})$ bits	$O(m \log\log n + occ)$	[12]
$O(n)$ bits	$O(m \log^2 n + occ \log n)$	[11]
	$O((m \log\log n + occ) \log^{\epsilon} n)$	[12]
	$O(m + (occ + \log^4 n \log\log n) \log^{\epsilon} n)$	[3]
$O(\lvert\Sigma\rvert n)$ words in avg	$O(m + occ)$	[13]
$O(\lvert\Sigma\rvert n)$ words	$O(m + occ)$ in avg	[13]

Space	$k = O(1)$	
$O(n \log^k n)$ words	$O(m + occ + \frac{1}{k!}(c \log n)^k \log\log n)$	[8]
$O(n \log^{k-1} n)$ words	$O(m + k^3 3^k occ + \frac{1}{k!}(c \log n)^k \log\log n)$	[3]
$O(n)$ words	$O(\min\{n, \lvert\Sigma\rvert^k m^{k+2}\} + occ)$	[6]
	$O((\lvert\Sigma\rvert m)^k \log n + occ)$	[11]
	$O(m + k^3 3^k occ + (c \log n)^{k(k+1)} \log\log n)$	[3]
	$O(\lvert\Sigma\rvert^k m^{k-1} \log n \log\log n + k^3 3^k occ)$	[4]
$O(n\sqrt{\log n})$ bits	$O((\lvert\Sigma\rvert m)^k \log\log n + occ)$	[12]
$O(n)$ bits	$O((\lvert\Sigma\rvert m)^k \log^2 n + occ \log n)$	[11]
	$O(((\lvert\Sigma\rvert m)^k \log\log n + occ) \log^{\epsilon} n)$	[12]
	$O(m + (k^3 3^k occ + (c \log n)^{k^2+2k} \log\log n) \log^{\epsilon} n)$	[3]
$O(\lvert\Sigma\rvert^k n \log^k n)$ words in avg	$O(m + occ)$	[13]
$O(\lvert\Sigma\rvert^k n \log^k n)$ words	$O(m + occ)$ in avg	[13]
$O(n \log^k n)$ words in avg	$O(3^k m^{k+1} + occ)$	[7]

Theorem 2 (Ferragina and Manzini, 2000 [9]; Grossi and Vitter [10]) *Given a compressed suffix array or an FM-index of size $O(n)$ bits, one can support exact (0-mismatch) matching in $O(m + occ \log^{\epsilon} n)$ time, where occ is the number of occurrences and ε is any positive constant smaller than or equal to 1.*

For inexact matching ($k \neq 0$), there are solutions whose indexes can help answer a k-mismatch/k-difference pattern query for any $k \geq 0$. Those indexes are created by augmenting the suffix tree and its variants. Theorems 3 to 7 summarize the current best results in such direction.

Theorem 3 (Chan, Lam, Sung, Tam, and Wong, 2006 [3]) *Given an index of size $O(n)$ words, one can support k-mismatch lookup in $O(m + occ + (c \log n)^{k(k+1)} \cdot \log\log n)$ time where c is a positive constant. For k-difference lookup, the term occ becomes $k^3 3^k occ$.*

Theorem 4 (Chan, Lam, Sung, Tam, and Wong, 2006 [3]) *Given an index of size $O(n)$ bits, one can support k-mismatch lookup in $O(m + (occ + (c \log n)^{k(k+2)} \cdot \log\log n) \log^{\epsilon} n)$ time where c is a positive constant and ε is any positive constant smaller than or equal to 1. For k-difference lookup, the term occ becomes $k^3 3^k occ$.*

Theorem 5 (Lam, Sung, and Wong, 2005 [12]) *Given an index of size $O(n\sqrt{\log n})$ bits, one can support k-mismatch/k-difference lookup in $O((\lvert\Sigma\rvert m)^k (k + \log\log n) + occ)$ time.*

Theorem 6 (Lam, Sung, and Wong, 2005 [12]) *Given an index of size $O(n)$ bits, one can support k-mismatch/k-difference lookup in $O(\log^{\epsilon}((\lvert\Sigma\rvert m)^k (k + \log\log n) + occ))$ time where ε is any positive constant smaller than or equal to 1.*

Theorem 7 (Chan, Lam, Sung, Tam, and Wong, 2006 [4]) *Given an index of size $O(n)$ words, one can support k-mismatch lookup in $O(\lvert\Sigma\rvert^k m^{k-1} \log n \log\log n +$*

occ) time. For k-difference lookup, the term occ becomes $k^3 3^k$ *occ.*

When k is given, one can create indexes whose sizes depend on k. Those solutions create the so-called k-error suffix tree and its variants. Theorems 8 to 11 summarize the current best results in this direction.

Theorem 8 (Maas and Nowak, 2005 [13]) *Given an index of size* $O(|\Sigma|^k n \log^k n)$ *words, one can support k-mismatch/k-difference lookup in* $O(m + occ)$ *expected time.*

Theorem 9 (Maas and Nowak, 2005 [13]) *Consider a uniformly and independently generated text of length n. One can construct an index of size* $O(|\Sigma|^k n \log^k n)$ *words on average, such that a k-mismatch/k-difference lookup query can be supported in* $O(m + occ)$ *worst case time.*

Theorem 10 (Chan, Lam, Sung, Tam, and Wong, 2006 [3]) *Given an index of size* $O(n \log^{k-h+1} n)$ *words where* $h \leq k$*, one can support k-mismatch lookup in* $O(m + occ + c^{k^2} \log^{\max\{kh,k+h\}} n \log\log n)$ *time where c is a positive constant. For k-difference lookup, the term occ becomes* $k^3 3^k$ *occ.*

Theorem 11 (Chan, Lam, Sung, Tam, and Wong, 2006 [4]) *Given an index of size* $O(n \log^{k-1} n)$ *words, one can support k-mismatch lookup in* $O(m + occ + \log^k n \cdot \log\log n)$ *time. For k-difference lookup, the term occ becomes* $k^3 3^k$ *occ.*

In addition, there are indexes which are efficient in practice for small k/m but give no worst case complexity guarantees. Those methods are based on filtration. The basic idea is to partition the pattern into short segments and locate those short segments in the text, allowing zero or a small number of errors. Those short segments help to identify candidate regions for the occurrences of the pattern. Finally, by verifying those candidate regions, one can recover all occurrences of the pattern. See [16] for a summary of those results. One of the best results based on filtration is stated in the following theorem.

Theorem 12 (Myers, 1994 [14]; Navarro and Baeza-Yates, 2000 [15]) *Consider an index of size* $O(n)$ *words. If* $k/m < 1 - O(1/\sqrt{\Sigma})$*, one can support a k-mismatch/k-difference search in* $O(n^\epsilon)$ *expected time where* ε *is a positive constant smaller than 1.*

All the above approaches either tried to index the strings with errors or are based on filtering. There are also solutions which use radically different approaches. For instance, there are solutions which transform approximate string searching into range queries in metric space [5].

Applications

Due to the advance in both internet and biological technologies, enormous text data is accumulated. For example, there is a 60G genomic sequence data in a gene bank. The data size is expected to grow exponentially.

To handle the huge data size, indexing techniques are vital to speed up the pattern matching queries. Moreover, exact pattern matching is no longer sufficient for both internet and biological data. For example, biological data usually contains a lot of differences due to experimental error and mutation and evolution. Therefore, approximate pattern matching becomes more appropriate. This gives the motivation for developing indexing techniques that allow pattern matching with errors.

Open Problems

The complexity for indexed approximate matching is still not fully understood. One would like to know the answers for a number of questions. For instance, one haves the following two questions. (1) Given a fixed index size of $O(n)$ words, what is the best time complexity of a k-mismatch/k-difference query? (2) If the k-mismatch/k-difference query time is fixed to $O(m + occ)$, what is the best space complexity of the index?

Cross References

► Text Indexing
► Two-Dimensional Pattern Indexing

Recommended Reading

1. Amir, A., Keselman, D., Landau, G.M., Lewenstein, M., Lewenstein, N., Rodeh, M.: Indexing and dictionary matching with one error. In: Proceedings of Workshop on Algorithms and Data Structures, 1999, pp. 181–192
2. Buchsbaum, A.L., Goodrich, M.T., Westbrook, J.R.: Range searching over tree cross products. In: Proceedings of European Symposium on Algorithms, 2000, pp. 120–131
3. Chan, H.-L., Lam, T.-W., Sung, W.-K., Tam, S.-L., Wong, S.-S.: A linear size index for approximate pattern matching. In: Proceedings of Symposium on Combinatorial Pattern Matching, 2006, pp. 49–59
4. Chan, H.-L., Lam, T.-W., Sung, W.-K., Tam, S.-L., Wong, S.-S.: Compressed indexes for approximate string matching. In: Proceedings of European Symposium on Algorithms, 2006, pp. 208–219
5. Navarro, G., Chávez, E.: A metric index for approximate string matching. Theor. Comput. Sci. **352**(1–3), 266–279 (2006)
6. Cobbs, A.: Fast approximate matching using suffix trees. In: Proceedings of Symposium on Combinatorial Pattern Matching, 1995, pp. 41–54
7. Coelho, L.P., Oliveira, A.L.: Dotted suffix trees: a structure for approximate text indexing. In: SPIRE, 2006, pp. 329–336

8. Cole, R., Gottlieb, L.A., Lewenstein, M.: Dictionary matching and indexing with errors and don't cares. In: Proceedings of Symposium on Theory of Computing, 2004, pp. 91–100
9. Ferragina, P., Manzini, G.: Opportunistic data structures with applications. In: Proceedings of Symposium on Foundations of Computer Science, 2000, pp. 390–398
10. Grossi, R., Vitter, J.S.: Compressed suffix arrays and suffix trees with applications to text indexing and string matching. In: Proceedings of Symposium on Theory of Computing, 2000, pp. 397–406
11. Huynh, T.N.D., Hon, W.K., Lam, T.W., Sung, W.K.: Approximate string matching using compressed suffix arrays. In: Proceedings of Symposium on Combinatorial Pattern Matching, 2004, pp. 434–444
12. Lam, T.W., Sung, W.K., Wong, S.S.: Improved approximate string matching using compressed suffix data structures. In: Proceedings of International Symposium on Algorithms and Computation, 2005, pp. 339–348
13. Maaß, M.G., Nowak, J.: Text indexing with errors. In: Proceedings of Symposium on Combinatorial Pattern Matching, 2005, pp. 21–32
14. Myers, E.G.: A sublinear algorithm for approximate keyword searching. Algorithmica **12**, 345–374 (1994)
15. Navarro, G., Baeza-Yates R.: A hybrid indexing method for approximate string matching. J. Discret. Algorithms **1**(1), 205–209 (2000)
16. Navarro, G., Baeza-Yates, R.A., Sutinen, E., Tarhio, J.: Indexing methods for approximate string matching. IEEE Data Eng. Bull. **24**(4), 19–27 (2001)
17. Weiner., P.: Linear Pattern Matching Algorithms. In: Proceedings of Symposium on Switching and Automata Theory, 1973, pp. 1–11

Inductive Inference

1983; Case, Smith

SANDRA ZILLES
Department of Computing Science, University of Alberta, Edmonton, AB, Canada

Keywords and Synonyms

Induction; Learning from examples

Problem Definition

The theory of inductive inference is concerned with the capabilities and limitations of machine learning. Here the learning machine, the concepts to be learned, as well as the hypothesis space are modeled in recursion theoretic terms, based on the framework of identification in the limit [1,8].

Formally, considering recursive functions (mapping natural numbers to natural numbers) as target concepts, a learner (inductive inference machine) is supposed to process, step by step, gradually growing segments of the graph of a target function. In each step, the learner outputs a program in some fixed programming system, where successful learning means that the sequence of programs returned in this process eventually stabilizes on some program actually computing the target function.

Case and Smith [2,3] have proposed several variants of this model in order to study the influence that certain constraints or relaxations may have on the capabilities of learners, thereby restricting (i) the number of mind changes (i. e., changes of output programs) a learner is allowed for in this process and (ii) the number of errors the program eventually hypothesized may have when compared to the target function.

One major result of studying the corresponding effects is a hierarchy of inference types culminating in a model general enough to allow for the identification of the whole class of recursive functions by a single inductive inference machine.

Notations

The target concepts for learning in the model discussed below are recursive functions [13] mapping natural numbers to natural numbers. Such functions, as well as partial recursive functions in general, are considered as computable in an arbitrary, but fixed acceptable numbering $\varphi = (\varphi_i)_{i \in \mathbb{N}}$. Here $\mathbb{N} = \{0, 1, 2, \dots\}$ denotes the set of all natural numbers. ϕ is interpreted as a programming system, where each $i \in \mathbb{N}$ is called a program for the partial recursive function φ_i.

Suppose f and g are partial recursive functions and $n \in \mathbb{N}$. Below $f =^n g$ is written if the set $\{x \in \mathbb{N} \mid f(x) \neq g(x)\}$ is of cardinality at most n. If the set $\{x \in \mathbb{N} \mid f(x) \neq g(x)\}$ is finite, this is denoted by $f =^* g$. One considers $*$ as a special symbol for which the $<$-relation is extended by $n < *$ for all $n \in \mathbb{N}$. For any recursive f and any $z \in \mathbb{N}$, let $f[z]$ denote $(z, (f(0), \dots, f(z)))$ for short.

For further basic recursion theoretic notions, the reader is referred to [13].

Learning Models

Case and Smith [3] build their theory upon the fundamental model of identification in the limit [1,8]. There a learner can be understood as an algorithmic device, called an inductive inference machine, which, given any 'graph segment' $f[z]$ as its input, returns a program $i \in \mathbb{N}$. Such a learner M identifies a recursive function f in the limit, if there is some $j \in \mathbb{N}$ such that

$$\varphi_j = f \text{ and } M(f[z]) = j \text{ for all but finitely many } z \in \mathbb{N} .$$

A class of recursive functions is learnable in the limit, if there is an inductive inference machine identifying each function in the class in the limit. Identification in the limit is called EX-identification, since a program for f is termed an *ex*planation for f.

For instance, the class of all primitive recursive functions is EX-identifiable, whereas the class of all recursive functions is not [8].

The central questions discussed by Case and Smith [3] are how the limitations of EX-learners are affected by posing certain requirements on the success criterion, concerning

- convergence criteria,
  - e. g., when restricting the number of permitted mind changes,
  - e. g., when relaxing the constraints on syntactical convergence of the sequence of programs returned in the learning process,
- accuracy,
  - e. g., when relaxing the number of permitted anomalies in the programs returned eventually.

Problem 1 *In which way do modifications of* EX*-identification in terms of accuracy and convergence criteria affect the capabilities of the corresponding learners?*

Problem 2 *In particular, if inaccuracies are permitted, can* EX*-learners always refute inaccurate hypotheses?*

Problem 3 *How much relaxation of the model of* EX*-identification is needed to achieve learnability of the full class of recursive functions?*

Key Results

Accuracy and Convergence Constraints

In order to systematically address these problems, Case and Smith [3] have defined inference types reflecting restrictions and relaxations of EX-identification as follows.

Definition 1 Suppose S is a class of recursive functions and $m, n \in \mathbb{N} \cup \{*\}$. S is EX_n^m-identifiable, if there is an inductive inference machine M, such that for any function $f \in S$ there is some $j \in \mathbb{N}$ satisfying

- $M(f[z]) = j$ for all but finitely many $z \in \mathbb{N}$,
- $\varphi_j =^m f$, and
- the cardinality of the set $\{z \in \mathbb{N} \mid M(f[z]) \neq M(f[z+1])\}$ is at most n.

EX_n^m denotes the set of all classes of recursive functions which are EX_n^m-identifiable.

Definition 2 Suppose S is a class of recursive functions and $m \in \mathbb{N} \cup \{*\}$. S is BC^m-identifiable, if there is an inductive inference machine M, which, for any function $f \in S$, satisfies

- $\varphi_{M(f[z])} =^m f$ for all but finitely many $z \in \mathbb{N}$.

BC^m denotes the set of all classes of recursive functions which are BC^m-identifiable. BC is short for behaviorally correct—the difference to EX-learning is that convergence of the sequence of programs returned by the learner is defined only in terms of semantics, no longer in terms of syntax.

The Impact of Accuracy and Convergence Constraints

In general, each permission of mind changes or anomalies increases the capabilities of learners; however mind changes cannot be traded in for anomalies or vice versa.

Theorem 1 *Let* $a, b, c, d \in \mathbb{N} \cup \{*\}$. *Then* $\mathrm{EX}_b^a \subseteq \mathrm{EX}_d^c$ *if and only if* $a \leq c$ *and* $b \leq d$.

Corollary 1 *For any* $m, n \in \mathbb{N}$ *the following inclusions hold.*

1. $\mathrm{EX}_n^m \subset \mathrm{EX}_n^{m+1} \subset \mathrm{EX}_n^*$.
2. $\mathrm{EX}_n^m \subset \mathrm{EX}_{n+1}^m \subset \mathrm{EX}_*^m$.

Theorem 2 *Let* $n \in \mathbb{N}$. *Then* $\mathrm{EX}_*^* \subset \mathrm{BC}^n \subset \mathrm{BC}^{n+1} \subset \mathrm{BC}^*$.

These results are essential concerning Problem 1.

Refutability

In particular, refutability demands in the sense that every incorrect hypothesis should be refutable (see [12]) are not applicable in the theory of inductive inference, see Problem 2.

Formally, Case and Smith [3] consider refutability as a property guaranteed by Popperian machines, the latter being defined as follows:

Definition 3 Suppose M is an inductive inference machine M. M is Popperian if, on any input, M returns a program of a recursive function.

Results thereon include:

Theorem 3 *There is an* EX*-identifiable class S of recursive functions for which there is no Popperian IIM witnessing its* EX*-identifiability.*

Corollary 2 *There is an* EX^1*-identifiable class S of recursive functions for which there is no Popperian IIM witnessing its* EX^1*-identifiability.*

Additionally, in EX^1-identification, Popper's refutability principle can not be applied even if it concerns only those hypotheses returned in the limit.

Learning All Recursive Functions

Since the results above yield a hierarchy of inference types with strictly growing collections of learnable classes, there is also an implicit answer to Problem 3: the class of recursive functions is neither in EX_n^m for any $m, n \in \mathbb{N} \cup \{*\}$ nor in BC^m for any $m \in \mathbb{N}$. In contrast to that, Case and Smith [3] prove

Theorem 4 *The class of all recursive functions is in* BC^*.

Applications

The work of Case and Smith [3] has been of high impact in learning theory.

A consequence of the discussion of anomalies has been that refutability principles in general do not hold for identification in the limit. This result has given rise to later studies on methods and techniques inductive inference machines might apply in order to discover their errors [6] and thus to further insights into the nature of inductive inference.

Concerning the study of mind change hierarchies, among others, their lifting to transfinite ordinal numbers [7] is a notable extension.

Moreover, the theory of learning as proposed by Case and Smith [3] has been applied for the development of the theory of identifying recursive [10] or recursively enumerable [9] languages.

Open Problems

Among the currently open problems in inductive inference, one key challenge is to find a reasonable notion of the complexity of learning problems (i. e., of classes of recursive functions) involving the run-time complexity of learners as well as the number of mind changes required to learn the functions in a class. In particular, special natural classes of functions should be analyzed in terms of such a complexity notion.

Though of course the hierarchies $\mathrm{EX}_0^m \subset \mathrm{EX}_1^m \subset \mathrm{EX}_2^m \subset \dots$ for any $m \in \mathbb{N}$ reflect some increase of complexity in that sense, a corresponding complexity notion would not address the aspect of run-time complexity of learners. Different complexity notions have been introduced, such as the so-called intrinsic complexity [5] (neglecting run-time complexity) and the 'measure under the curve' [4] (respecting the number of examples required, but neglecting the number of mind changes). In particular, for learning deterministic finite automata, different notions of run-time complexity have been discussed [11].

However, the definition of a more capacious complexity notion remains an open issue.

Cross References

▶ PAC Learning

Recommended Reading

1. Blum, L., Blum, M.: Toward a mathematical theory of inductive inference. Inform. Control **28**(2), 125–155 (1975)
2. Case, J., Smith, C.H.: Anomaly hierarchies of mechanized inductive inference. In: Proceedings of the 10th Symposium on the Theory of Computing, pp. 314–319. ACM, New York (1978)
3. Case, J., Smith, C.H.: Comparison of Identification Criteria for Machine Inductive Inference. Theor. Comput. Sci. **25**(2), 193–220 (1983)
4. Daley, R.P., Smith, C.H.: On the Complexity of Inductive Inference. Inform. Control **69**(1–3), 12–40 (1986)
5. Freivalds, R., Kinber, E., Smith, C.H.: On the Intrinsic Complexity of Learning. Inform. Comput. **118**(2), 208–226 (1995)
6. Freivalds, R., Kinber, E., Wiehagen, R.: How inductive inference strategies discover their errors. Inform. Comput. **123**(1), 64–71 (1995)
7. Freivalds, R., Smith, C.H.: On the Role of Procrastination in Machine Learning. Inform. Comput. **107**(2), 237–271 (1993)
8. Gold, E.M.: Language identification in the limit. Inform. Control **10**(5), 447–474 (1967)
9. Kinber, E.B., Stephan, F.: Language Learning from Texts: Mindchanges, Limited Memory, and Monotonicity. Inform. Comput. **123**(2), 224–241 (1995)
10. Lange, S., Grieser, G., Zeugmann, T.: Inductive inference of approximations for recursive concepts. Theor. Comput. Sci. **348**(1), 15–40 (2005)
11. Pitt, L.: Inductive inference, DFAs, and computational complexity. In: Analogical and Inductive Inference, 2nd International Workshop, Reinhardsbrunn Castle, GDR. Lecture Notes in Computer Science, vol. 397, pp. 18–44. Springer, Berlin (1989)
12. Popper, K.: The Logic of Scientific Discovery. Harper & Row, New York (1959)
13. Rogers, H.: Theory of Recursive Functions and Effective Computability. McGraw-Hill, New York (1967)

I/O-model

1988; Aggarwal, Vitter

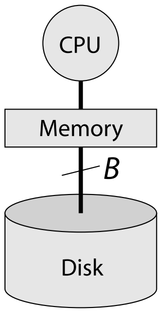
NORBERT ZEH
Faculty of Computer Science, Dalhousie University, Halifax, NS, Canada

Keywords and Synonyms

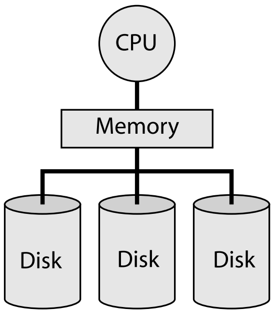
External-memory model; Disk access model (DAM)

Definition

The Input/Output model (I/O-model) [1] views the computer as consisting of a *processor*, *internal memory* (RAM), and *external memory* (disk). See Fig. 1. The internal mem-



I/O-model, Figure 1
The I/O-model



I/O-model, Figure 2
The parallel disk model

ory is of limited size, large enough to hold M data items. The external memory is of conceptually unlimited size and is divided into *blocks* of B consecutive data items. All computation has to happen on data in internal memory. Data is brought into internal memory and written back to external memory using *I/O-operations* (I/Os), which are performed explicitly by the algorithm. Each such operation reads or writes one block of data from or to external memory. The complexity of an algorithm in this model is the number of I/Os it performs.

The *parallel disk model* (PDM) [10] is an extension of the I/O-model that allows the external memory to consist of $D \geq 1$ parallel disks. See Fig. 2. In this model, a single I/O-operation is capable of reading or writing up to D independent blocks, as long as each of them is stored on a different disk.

Key Results

A few complexity bounds are of importance to virtually every I/O-efficient algorithm or data structure. The *searching bound* of $\Theta(\log_B n)$ I/Os, which can be achieved using a B-tree [4], is the cost of searching for an element in an ordered collection of n elements, using comparisons only. It is thus the equivalent of the $\Theta(\log n)$ searching bound in internal memory.

Scanning a list of n consecutive data items obviously takes $\lceil n/B \rceil$ I/Os or, in the PDM, $\lceil n/DB \rceil$ I/Os. This *scanning bound* is usually referred to as a "linear number of I/Os" because it is the equivalent of the $O(n)$ time bound required to do the same in internal memory.

The *sorting bound* of $\text{sort}(n) = \Theta((n/B)\log_{M/B}(n/B))$ I/Os denotes the cost of sorting n elements using comparisons only. It is thus the equivalent of the $\Theta(n \log n)$ sorting bound in internal memory. In the PDM, the sorting bound becomes $\Theta((n/DB)\log_{M/B}(n/B))$. This bound can be achieved using a range of sorting algorithms, including external merge sort [1,6] and distribution sort [1,5].

Arguably the most interesting bound is the *permutation bound*, that is, the cost of rearranging n elements in a given order, which is $\Theta(\min(\text{sort}(n), n))$ [1] or, in the PDM, $\Theta(\min(\text{sort}(n), n/D))$ [10]. For all practical purposes, this is the same as the sorting bound. Note the contrast to internal memory where, up to constant factors, permuting has the same cost as a linear scan. Since almost all non-trivial algorithmic problems include a permutation problem, this implies that only exceptionally simple problems can be solved in $O(\text{scan}(n))$ I/Os; most problems have an $\Omega(\text{perm}(n))$, that is, essentially an $\Omega(\text{sort}(n))$ lower bound. Therefore, while internal-memory algorithms aiming for linear time have to carefully avoid the use of sorting as a tool, external-memory algorithms can sort without fear of significantly exceeding the lower bound. This makes the design of I/O-optimal algorithms potentially easier than the design of optimal internal-memory algorithms. It is, however, counterbalanced by the fact that, unlike in internal memory, the sorting bound is *not* equal to n times the searching bound, which implies that algorithms based on querying a tree-based search structure $O(n)$ times usually do not translate into I/O-efficient algorithms. *Buffer trees* [3] achieve an amortized search bound of $O((1/B)\log_{M/B}(N/B))$ I/O, but can be used only if the entire update and query sequence is known in advance and thus provide only a limited solution to this problem.

Apart from these fundamental results, there exist a wide range of interesting techniques, particularly for solving geometric and graph problems. For surveys, refer to [2,9].

Applications

Modern computers are equipped with memory hierarchies consisting of several levels of cache memory, main memory (RAM), and disk(s). Access latencies increase with the distance from the processor, as do the sizes of the memory levels. To amortize these increasing access latencies, data are transferred between different levels of cache in blocks of consecutive data items. As a result, the cost of a memory access depends on the level in the memory hierarchy currently holding the data item—the difference in access latency between L1 cache and disk is about 10^6—and the cost of a sequence of accesses to data items stored at the same level depends on the number of blocks over which these items are distributed.

Traditionally, algorithms have been designed to minimize the number of computation steps; the access locality necessary to solve a problem using few data transfers between memory levels has been largely ignored. Hence, the designed algorithms work well on data sets of moderate size, but do not take noticeable advantage of cache memory and usually break down completely in out-of-core computations. Since the difference in access latencies is largest between main memory and disk, the I/O-model focuses on minimizing this I/O-bottleneck. This two-level view of the memory hierarchy keeps the model simple and useful for analyzing sophisticated algorithms, while providing a good prediction of their practical performance.

Much effort has been made already to translate provably I/O-efficient algorithms into highly efficient implementations. Examples include TPIE [8] and STXXL [7], two libraries that aim to provide highly optimized and powerful primitives for the implementation of I/O-efficient algorithms. In spite of these efforts, a significant gap between the theory and practice of I/O-efficient algorithms remains (see next section).

Open Problems

There are a substantial number of open problems in the area of I/O-efficient algorithms. The most important ones concern graph and geometric problems.

Traditional graph algorithms usually use a well-organized graph traversal such as depth-first search or breadth-first search to gain information about the structure of the graph and then use this information to solve the problem at hand. In the I/O-model, no I/O-efficient depth-first search algorithm is known and for breadth-first search and shortest paths, progress has been made only recently on undirected graphs. For directed graphs, even such simple problems as deciding whether there exists a directed path between two vertices are currently open. The main research focus in this area is therefore to either develop I/O-efficient general traversal algorithms or to continue the current strategy of devising graph algorithms that depart from traditional traversal-based approaches.

Techniques for solving geometric problems I/O-efficiently are much better understood than is the case for graph algorithms, at least in two dimensions. Nevertheless, there are a few important frontiers that remain. Arguably the most important one is the development of I/O-efficient algorithms and data structures for higher-dimensional geometric problems. Motivated by database applications, higher-dimensional range searching is one of the problems to be studied in this context. Little work has been done in the past on solving proximity problems, which pose another frontier currently explored by researchers in the field. Motivated by the need for such structures in a range of application areas and in particular in geographic information systems, there has been some recent focus on the development of multifunctional data structures, that is, structures that can answer different types of queries efficiently. Most existing structures are carefully tuned to efficiently support *one* particular type of query.

For both, I/O-efficient graph algorithms and computational geometry, there is a substantial gap between the obtained theoretical results and what is known to be practical, even though more experimental work has been done on geometric algorithms than on graph algorithms. Thus, if I/O-efficient algorithms in these areas are to have any practical impact, increased efforts are needed to bridge this gap by developing practically I/O-efficient algorithms that are still *provably* efficient.

Cross References

For details on ► External Sorting and Permuting, please refer to the corresponding entry. Details on one- and higher-dimensional searching are provided in the entries on ► B-trees and ► R-trees. The reader interested in algorithms that focus on efficiency at all levels of the memory hierarchy should consult the entry on the ► Cache-Oblivious Model.

Recommended Reading

1. Aggarwal, A., Vitter, J.S.: The input/output complexity of sorting and related problems. Commun. ACM **31**(9), 1116–1127 (1988)
2. Arge, L.: External memory data structures. In: Abello, J., Pardalos, P.M., Resende, M.G.C. (eds.) Handbook of Massive Data Sets, pp. 313–357. Kluwer Academic Publishers, Dordrecht (2002)
3. Arge, L.: The buffer tree: A technique for designing batched external data structures. Algorithmica **37**(1), 1–24 (2003)

4. Bayer, R., McCreight, E.: Organization of large ordered indexes. Acta Inform. **1**, 173–189 (1972)
5. Nodine, M.H., Vitter, J.S.: Deterministic distribution sort in shared and distributed memory multiprocessors. In: Proceedings of the 5th Annual ACM Symposium on Parallel Algorithms and Architectures, pp. 120–129. Velen, June/July 1993
6. Nodine, M.H., Vitter, J.S.: Greed Sort: An optimal sorting algorithm for multiple disks. J. ACM **42**(4), 919–933 (1995)
7. STXXL: C++ Standard Library for Extra Large Data Sets. http://stxxl.sourceforge.net. Accessed: 15 March 2008
8. TPIE — A Transparent Parallel I/O-Environment. http://www.cs.duke.edu/TPIE. Accessed: 15 March 2008
9. Vitter, J.S.: External memory algorithms and data structures: Dealing with massive data. ACM Comput. Surv. **33**(2), 209–271 (2001)
10. Vitter, J.S., Shriver, E.A.M.: Algorithms for parallel memory I: Two-level memories. Algorithmica **12**(2–3), 110–147 (1994)

K

Kinetic Data Structures

1999; Basch, Guibas, Hershberger

Bettina Speckmann
Department of Mathematics and Computer Science,
Technical University of Eindhoven,
Eindhoven, The Netherlands

Problem Definition

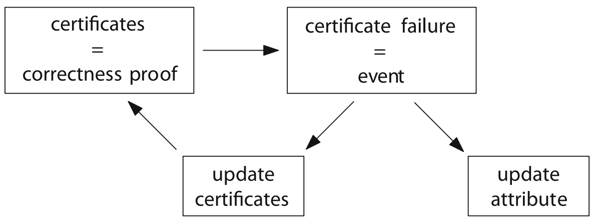
Many application areas of algorithms research involve objects in motion. Virtual reality, simulation, air-traffic control, and mobile communication systems are just some examples. Algorithms that deal with objects in motion traditionally discretize the time axis and compute or update their structures based on the position of the objects at every time step. If all objects move continuously then in general their configuration does not change significantly between time steps—the objects exhibit *spatial* and *temporal coherence*. Although *time-discretization* methods can exploit spatial and temporal coherence they have the disadvantage that it is nearly impossible to choose the perfect time step. If the distance between successive steps is too large, then important interactions might be missed, if it is too small, then unnecessary computations will slow down the simulation. Even if the time step is chosen just right, this is not always a satisfactory solution: some objects may have moved only slightly and in such a way that the overall data structure is not influenced.

One would like to use the temporal coherence to detect precisely those points in time when there is an actual change in the structure. The *kinetic data structure* (KDS) framework, introduced by Basch et al. in their seminal paper [2], does exactly that: by maintaining not only the structure itself, but also some additional information, they can determine when the structure will undergo a "real" (combinatorial) change.

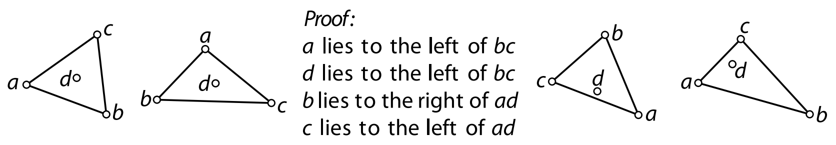
Key Results

A kinetic data structure is designed to maintain or monitor a discrete attribute of a set of moving objects, for example, the convex hull or the closest pair. The basic idea is, that although all objects move continuously, there are only certain discrete moments in time when the combinatorial structure of the attribute changes (in the earlier examples, the ordered set of convex-hull vertices or the pair that is closest, respectively). A KDS therefore contains a set of *certificates* that constitutes a proof of the property of interest. Certificates are generally simple inequalities that assert facts like "point c is on the left of the directed line through points a and b." These certificates are inserted in a priority queue (*event queue*) based on their time of expiration. The KDS then performs an event-driven simulation of the motion of the objects, updating the structure whenever an *event* happens, that is, when a certificate fails (see Fig. 1). It is part of the art of designing efficient kinetic data structures to find a small set of simple and easily updatable certificates that serve as a proof of the property one wishes to maintain.

A KDS assumes that each object has a known motion trajectory or *flight plan*, which may be subject to restrictions to make analysis tractable. Two common restrictions would be translation along paths parametrized by polynomials of fixed degree d, or translation and rotation described by algebraic curves. Furthermore, certificates are generally simple algebraic equations, which implies that



Kinetic Data Structures, Figure 1
The basic structure of an event based simulation with a KDS



Proof:
a lies to the left of *bc*
d lies to the left of *bc*
b lies to the right of *ad*
c lies to the left of *ad*

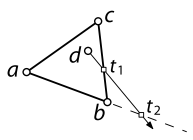
Kinetic Data Structures, Figure 2
Equivalent convex hull configurations (*left* and *right*), a proof that *a*, *b*, and *c* form the convex hull of *S* (*center*)

the failure time of a certificate can be computed as the next largest root of an algebraic expression. An important aspect of kinetic data structures is their on-line character: although the positions and motions (flight plans) of the objects are known at all times, they are not necessarily known far in advance. In particular, any object can change its flight plan at any time. A good KDS should be able to handle such changes in flight plans efficiently.

A detailed introduction to kinetic data structures can be found in Basch's Ph. D. thesis [1] or in the surveys by Guibas [3,4]. In the following the principles behind kinetic data structures are illustrated by an easy example.

Consider a KDS that maintains the convex hull of a set S of four points a, b, c, and d as depicted in Fig. 2. A set of four simple certificates is sufficient to certify that a, b, and c form indeed the convex hull of S (see Fig. 2 center). This implies, that the convex hull of S will not change under any motion of the points that does not lead to a violation of these certificates. To put it differently, if the points move along trajectories that move them between the configurations depicted in Fig. 2 without the point d ever appearing on the convex hull, then the KDS in principle does not have to process a single event.

Now consider a setting in which the points a, b, and c are stationary and the point d moves along a linear trajectory (Fig. 3 left). Here the KDS has exactly two events to process. At time t_1 the certificate "d is to the left of bc" fails as the point d appears on the convex hull. In this easy setting, only the failed certificate is replaced by "d is to the *right* of bc" with failure time "never", generally processing an event would lead to the scheduling and descheduling of several events from the event queue. Finally at time t_2 the certificates "b is to the right of ad" fails as the point b ceases



Certificate	Failure time
a lies to the left of *bc*	never
d lies to the left of *bc*	t_1
b lies to the right of *ad*	t_2
c lies to the left of *ad*	never

Kinetic Data Structures, Figure 3
Certificate structure for points *a*, *b*, and *c* being stationary and point *d* moving along a straight line

to be on the convex hull and is replaced by "b is to the *left* of ad" with failure time "never."

Kinetic data structures and their accompanying maintenance algorithms can be evaluated and compared with respect to four desired characteristics.

Responsiveness. One of the most important performance measures for a KDS is the time needed to update the attribute and to repair the certificate set when a certificate fails. A KDS is called *responsive* if this update time is "small", that is, polylogarithmic.

Compactness. A KDS is called *compact* if the number of certificates is near-linear in the total number of objects. Note that this is not necessarily the same as the amount of storage the entire structure needs.

Locality. A KDS is called *local* if every object is involved in only a small number of certificates (again, "small" translates to polylogarithmic). This is important whenever an object changes its flight plane, because one has to recompute the failure times of all certificates this object is involved in, and update the event queue accordingly. Note that a local KDS is always compact, but that the reverse is not necessarily true.

Efficiency. A certificate failure does not automatically imply a change in the attribute that is being maintained, it can also be an *internal event*, that is, a change in some auxiliary structure that the KDS maintains. A KDS is called *efficient* if the worst-case number of events handled by the data structure for a given motion is small compared to the number of combinatorial changes of the attribute (*external events*) that must be handled for that motion.

Applications

The paper by Basch et al. [2] sparked a large amount of research activities and over the last years kinetic data structures have been used to solve various dynamic computational geometry problems. A number of papers deal foremost with the maintenance of discrete attributes for sets of moving points, like the closest pair, width and diameter, clusters, minimum spanning trees, or the constrained Delaunay triangulation. Motivated by ad hoc mobile networks, there have also been a number of papers that show how to maintain the connected components in a set of moving regions in the plane. Major research efforts have also been seen in the study of kinetic binary space partitions (BSPs) and kinetic kd-trees for various objects. Finally, there are several papers that develop KDSs for collision detection in the plane and in three dimensions. A detailed discussion and an extensive list of references can be found in the survey by Guibas [4].

Cross References

▶ Fully Dynamic Minimum Spanning Trees
▶ Minimum Geometric Spanning Trees

Recommended Reading

1. Basch, J.: Kinetic Data Structures. Ph. D. thesis, Stanford University (1999)
2. Basch, J., Guibas, L., Hershberger, J.: Data structures for mobile data. J. Algorithms **31**, 1–28 (1999)
3. Guibas, L.: Kinetic data structures: A state of the art report. In: Proc. 3rd Workshop on Algorithmic Foundations of Robotics, pp. 191–209 (1998)
4. Guibas, L.: Modeling Motion. In: Goodman, J., O'Rourke, J.: (eds), Handbook of Discrete and Computational Geometry. CRC Press, 2nd ed. (2004)

Knapsack

1975; Ibarra, Kim

Hans Kellerer
Department of Computer Science, University of Graz, Graz, Austria

Keywords and Synonyms

Approximation algorithm; Fully polynomial time approximation scheme (FPTAS)

Problem Definition

For a given set of items $N = \{1, \dots, n\}$ with nonnegative integer weights w_j and profits p_j, $j = 1, \dots, n$, and a knapsack of capacity c, the *knapsack problem* (KP) is to select a subset of the items such that the total profit of the selected items is maximized and the corresponding total weight does not exceed the knapsack capacity c.

Alternatively, a knapsack problem can be formulated as a solution of the following linear integer programming formulation:

$$(KP) \qquad \text{maximize} \sum_{j=1}^{n} p_j x_j \tag{1}$$

$$\text{subject to} \sum_{j=1}^{n} w_j x_j \le c\,, \tag{2}$$

$$x_j \in \{0, 1\}, \quad j = 1, \dots, n\,. \tag{3}$$

The knapsack problem is the simplest non-trivial integer programming model having binary variables, only a single constraint and only positive coefficients. A large number of theoretical and practical papers has been published on this problem and its extensions. An extensive overview can be found in the books by Kellerer, Pferschy and Pisinger [2] or Martello and Toth [7].

Adding the integrality condition (3) to the simple linear program (1)-(2) already puts (KP) into the class of $\mathcal{NP}$-hard problems. Thus, (KP) admits no polynomial time algorithms unless $\mathcal{P} = \mathcal{NP}$ holds.

Therefore, this entry will focus on approximation algorithms for (KP). A common method to judge the quality of an approximation algorithm is its worst-case performance. For a given instance I define by $z^*(I)$ the optimal solution value of (KP) and by $z^H(I)$ the corresponding solution value of a heuristic H. For $\varepsilon \in [0, 1[$ a heuristic H is called a $(1 - \varepsilon)$–*approximation algorithm* for (KP) if for any instance I

$$z^H(I) \ge (1 - \varepsilon) z^*(I)$$

holds. Given a parameter ε, a heuristic H is called a *fully polynomial approximation scheme*, or an FTPAS, if H is a $(1 - \varepsilon)$–approximation algorithm for (KP) for any $\varepsilon \in [0, 1[$, and its running time is polynomial both in the length of the encoded input n and $1/\varepsilon$. The first FTPAS for (KP) was suggested by Ibarra and Kim [1] in 1975. It was among the early FPTASes for discrete optimization problems. It will be described in detail in the following.

Key Results

(KP) can be solved in pseudopolynomial time by a simple dynamic programming algorithm. One possible variant is the so-called *dynamic programming by profits* (DP-Profits). The main idea of DP-Profits is to reach every possible total profit value with a subset of items of minimal total weight. Clearly, the highest total profit value, which can be reached by a subset of weight not greater than the capacity c, will be an optimal solution.

Let $y_j(q)$ denote the minimal weight of a subset of items from $\{1, \dots, j\}$ with total profit equal to q. To bound the length of every array y_j an upper bound u on the optimal solution value has to be computed. An obvious possibility would be to use the upper bound $U_{\text{LP}} = \lfloor z^{LP} \rfloor$ from the solution z^{LP} of the LP-relaxation of (KP) and set

$U := U_{\text{LP}}$. It can be shown that U_{LP} is at most twice as large as the optimal solution value z^*. Initializing $y_0(0) := 0$ and $y_0(q) := c + 1$ for $q = 1, \ldots, U$, all other values can be computed for $j = 1, \ldots, n$ and $q = 0, \ldots, U$ by using the recursion

$$y_j(q) := \begin{cases} y_{j-1}(q) & \text{if } q < p_j\,, \\ \min\{y_{j-1}(q),\ y_{j-1}(q - p_j) + w_j\} & \text{if } q \geq p_j\,. \end{cases}$$

The optimal solution value is given by $\max\{q \mid y_n(q) \leq c\}$ and the running time of DP-Profits is bounded by $O(nU)$.

Theorem 1 (Ibarra, Kim) *There is an FTPAS for (KP) which runs in* $O(n \log n + n/\varepsilon^2)$ *time.*

Proof The FTPAS is based on appropriate *scaling* of the profit values p_j and then running DP-Profits with the scaled profit values. Scaling means here that the given profit values p_j are replaced by new profits $\tilde{p}_j$ such that $\tilde{p}_j := \left\lfloor \frac{p_j}{K} \right\rfloor$ for an appropriate chosen constant K.

This scaling can be seen as a partitioning of the profit range into intervals of length K with starting points $0, K, 2K, \ldots$. Naturally, for every profit value p_j there is some integer value $i \geq 0$ such that p_j falls into the interval $[iK, (i+1)K[$. The scaling procedure generates for every p_j the value $\tilde{p}_j$ as the corresponding index i of the lower interval bound iK.

Running DP-Profits yields a solution set $\tilde{X}$ for the scaled items which will usually be different from the original optimal solution set X^*. Evaluating the original profits of item set $\tilde{X}$ yields the approximate solution value z^H. The difference between z^H and the optimal solution value can be bounded as follows.

$$z^H \geq \sum_{j \in \tilde{X}} K \left\lfloor \frac{p_j}{K} \right\rfloor \geq \sum_{j \in X^*} K \left\lfloor \frac{p_j}{K} \right\rfloor \geq \sum_{j \in X^*} K \left(\frac{p_j}{K} - 1 \right) = z^* - |X^*| K.$$

To get the desired performance guarantee of $1 - \varepsilon$ it is sufficient to have $\frac{z^* - z^H}{z^*} \leq \frac{|X^*|K}{z^*} \leq \varepsilon$.

To ensure this K has to be choosen such that

$$K \leq \frac{\varepsilon z^*}{|X^*|}\,. \tag{4}$$

Since $n \geq |X^*|$ and $U_{\text{LP}}/2 \leq z^*$ choosing $K := \frac{\varepsilon U_{\text{LP}}}{2n}$ satisfies condition (4) and thus guarantees the performance ratio of $1 - \varepsilon$. Substituting U in the $O(nU)$ bound for DP-Profits by U/K yields an overall running time of $O(n^2 \varepsilon)$.

A further improvement in the running time is obtained in the following way. Separate the items into *small* items (having profit $\leq \frac{\varepsilon}{2} U_{\text{LP}}$) and *large* items (having profit $> \frac{\varepsilon}{2} U_{\text{LP}}$). Then, perform DP-Profits for the scaled large items only. To each entry q of the obtained dynamic programming array with corresponding weight $y(q)$ the small items are added to a knapsack with residual capacity $c - y(q)$ in a greedy way. The small items shall be sorted in non-increasing order of their profit to weight ratio. Out of the resulting combined profit values, the highest one is selected. Since every optimal solution contains at most $2/\varepsilon$ large items, $|X^*|$ can be replaced in (4) by $2/\varepsilon$ which results in an overall running time $O(n \log n + n/\varepsilon^2)$. The memory requirement of the algorithm is $O(n + 1/\varepsilon^3)$. □

Two important approximation schemes with advanced treatment of items and algorithmic fine tuning were presented some years later. The classical paper by Lawler [5] gives a refined scaling resp. partitioning of the items and several other algorithmic improvements which results in a running time $O(n \log(1/\varepsilon) + 1/\varepsilon^4)$. A second paper by Magazine and Oguz [6] contains among other features a partitioning and recombination technique to reduce the space requirements of the dynamic programming procedure. The fastest algorithm is due to Kellerer and Pferschy [3,4] with running time $O(n \min\{\log n, \log(1/\varepsilon)\} + 1/\varepsilon^2 \log(1/\varepsilon) \cdot \min\{n, 1/\varepsilon \log(1/\varepsilon)\})$ and space requirement $O(n + 1/\varepsilon^2)$.

Applications

(KP) is one the classical problems in combinatorial optimization. Since (KP) has this simple structure and since there are efficient algorithms for solving it, many solution methods of more complex problems employ the knapsack problem (sometimes iteratively) as a subproblem.

A straightforward interpretation of (KP) is an investment problem. A wealthy individual or institutional investor has a certain amount of money c available which he wants to put into profitable business projects. As a basis for his decisions he compiles a long list of possible investments including for every investment the required amount w_j and the expected net return p_j over a fixed period. The aspect of risk is not explicitly taken into account here. Obviously, the combination of the binary decisions for every investment such that the overall return on investment is as large as possible can be formulated by (KP).

One may also view the (KP) as a "cutting" problem. Assume that a sawmill has to cut a log into shorter pieces. The pieces must however be cut into some predefined standard-lengths w_j, where each length has an associated selling price p_j. In order to maximize the profit of the log, the sawmill can formulate the problem as a (KP) where the length of the log defines the capacity c.

Among the wide range of "real world" applications shall be mentioned two-dimensional cutting problems, column generation, separation of cover inequalities, financial decision problems, asset-backed securitization, scheduling problems, knapsack cryptosystems and most recent combinatorial auctions. For a survey on applications of knapsack problems the reader is referred to [2].

Recommended Reading

1. Ibarra, O.H., Kim, C.E.: Fast approximation algorithms for the knapsack and sum of subset problem. J. ACM **22**, 463–468 (1975)
2. Kellerer, H., Pisinger, D., Pferschy U.: Knapsack Problems. Springer, Berlin (2004)
3. Kellerer, H., Pferschy, U.: A new fully polynomial time approximation scheme for the knapsack problem. J. Comb. Optim. **3**, 59–71 (1999)
4. Kellerer, H., Pferschy, U.: Improved dynamic programming in connection with an FPTAS for the knapsack problem. J. Comb. Optim. **8**, 5-11 (2004)
5. Lawler, E.L.: Fast approximation algorithms for knapsack problems. Math. Oper. Res. **4**, 339–356 (1979)
6. Magazine, M.J., Oguz, O.: A fully polynomial approximation algorithm for the 0–1 knapsack problem. Eur. J. Oper. Res. **8**, 270–273 (1981)
7. Martello, S., Toth, P. Knapsack Problems: Algorithms and Computer Implementations. Wiley, Chichester (1990)

L

Learning with the Aid of an Oracle

1996; Bshouty, Cleve, Gavaldà, Kannan, Tamon

Christino Tamon
Department of Mathematics and Computer Science,
Clarkson University,
Potsdam, NY, USA

Keywords and Synonyms

Oracles and queries that are sufficient for exact learning

Problem Definition

In the exact learning model of Angluin [1], a learning algorithm A must discover an unknown function $\mathsf{f}\colon \{0,1\}^n \to \{0,1\}$ that is a member of a known class C of Boolean functions. The learning algorithm can make at least one of the following types of queries about f:

- Equivalence query $\mathsf{EQ_f(g)}$, for a candidate function g: The reply is either "yes", if $\mathsf{g} \Leftrightarrow \mathsf{f}$, or a counterexample a with $\mathsf{g}(a) \neq \mathsf{f}(a)$, otherwise.
- Membership query $\mathsf{MQ_f}(a)$, for some $a \in \{0,1\}^n$: The reply is the Boolean value $\mathsf{f}(a)$.
- Subset query $\mathsf{SubQ_f(g)}$, for a candidate function g: The reply is "yes", if $\mathsf{g} \Rightarrow \mathsf{f}$, or a counterexample a with $\mathsf{f}(a) < \mathsf{g}(a)$, otherwise.
- Superset query $\mathsf{SupQ_f(g)}$, for a candidate function g: The reply is "yes", if $\mathsf{f} \Rightarrow \mathsf{g}$, or a counterexample a with $\mathsf{g}(a) < \mathsf{f}(a)$, otherwise.

A Disjunctive Normal Formula (DNF) is a depth-2 OR-AND circuit whose size is given by the number of its AND gates. Likewise, a Conjunctive Normal Formula (CNF) is a depth-2 AND-OR circuit whose size is given by the number of its OR gates. Any Boolean function can be represented as both a DNF or a CNF formula. A k-DNF is a DNF where each AND gate has a fan-in of at most k; similarly, it is possible to define k-CNF.

Problem

For a given class C of Boolean functions, such as polynomial-size Boolean circuits or Disjunctive Normal Form (DNF) formulas, the goal is to design polynomial-time learning algorithms for any unknown $\mathsf{f} \in \mathsf{C}$ and ask a polynomial number of queries. The output of the learning algorithm should be a function g of polynomial size satisfying $g \Leftrightarrow f$. The polynomial functions bounding the running time, query complexity, and output size are defined in terms of the number of inputs n and the size of the smallest representation (Boolean circuit or DNF) of the unknown function f

Key Results

One of the main results proved in [4] is that Boolean circuits and Disjunctive Normal Formulas are exactly learnable using equivalence queries and access to an NP oracle.

Theorem 1 *The following tasks can be accomplished with probabilistic polynomial-time algorithms that have access to an* NP *oracle and make polynomially many equivalence queries:*

- *Learning* DNF *formulas of size s using equivalence queries that are depth-3* AND-OR-AND *formulas of size $O(sn^2/log^2 n)$.*
- *Learning Boolean circuits of size s using equivalence queries that are circuits of size $O(sn + n\log n)$.*

The idea behind this result is simple. Any class C of Boolean functions is exactly learnable with equivalence queries using the Halving algorithm of Littlestone [10]. This algorithm asks equivalence queries that are the *majority* of candidate functions from C. These are functions in C that are consistent with the counterexamples obtained so far by the learning algorithm. Since each such majority query eliminates at least half of the candidate functions, $\log_2 |\mathsf{C}|$ equivalence queries are sufficient to learn any function in C. A problem with using the Halving algorithm here is that the majority query has exponential size. But, it can be shown that a majority of a polynomial number of uniformly random candidate functions is a good enough approximator to the majority of all candidate functions. Moreover, with access to an NP oracle, there is a randomized polynomial time algorithm for generating random uniform candidate functions due to Jerrum, Valiant, and Vazirani [6]. This yields the result.

The next observation is that subset and superset queries are apparently powerful enough to simulate both equivalence queries and the NP oracle. This is easy to see since the tautology test $\mathsf{g} \Leftrightarrow 1$ is equivalent to $\mathsf{SubQ_f}(\overline{\mathsf{g}}) \wedge \mathsf{SubQ_f}(\mathsf{g})$, for any unknown function f; and, $\mathsf{EQ_f}(\mathsf{g})$ is equivalent to $\mathsf{SubQ_f}(\mathsf{g}) \wedge \mathsf{SupQ_f}(\mathsf{g})$. Thus, the following generalization of Theorem 1 is obtained.

Theorem 2 *The following tasks can be accomplished with probabilistic polynomial-time algorithms that make polynomially many subset and superset queries:*
- *Learning* DNF *formulas of size s using equivalence queries that are depth-3* AND-OR-AND *formulas of size $O(sn^2/log^2 n)$.*
- *Learning Boolean circuits of size s using equivalence queries that are circuits of size $O(sn + n \log n)$.*

Stronger deterministic results are obtained by allowing more powerful complexity-theoretic oracles. The first of these results employ techniques developed by Sipser and Stockmeyer [11,12].

Theorem 3 *The following tasks can be accomplished with deterministic polynomial-time algorithms that have access to an Σ_3^p oracle and make polynomially many equivalence queries:*
- *Learning* DNF *formulas of size s using equivalence queries that are depth-3* AND-OR-AND *formulas of size $O(sn^2/log^2 n)$.*
- *Learning Boolean circuits of size s using equivalence queries that are circuits of size $O(sn + n \log n)$.*

In the following result, C is an infinite class of functions containing functions of the form $\mathsf{f}\colon \{0,1\}^\star \to \{0,1\}$. The class C is p-evaluatable if the following tasks can be performed in polynomial time:
- Given y, is y a valid representation for any function $\mathsf{f}_y \in \mathsf{C}$?
- Given a valid representation y and $x \in \{0,1\}^\star$, is $\mathsf{f}_y(x) = 1$?

Theorem 4 *Let* C *be any p-evaluatable class. The following statements are equivalent:*
- C *is learnable from polynomially many equivalence queries of polynomial size (and unlimited computational power).*
- C *is learnable in deterministic polynomial time with equivalence queries and access to a Σ_5^p oracle.*

For exact learning with membership queries, the following results are proved.

Theorem 5 *The following tasks can be accomplished with deterministic polynomial-time algorithms that have access to an* NP *oracle and make polynomially many membership queries (in n,* DNF *and* CNF *sizes of* f*, where* f *is the unknown function):*
- *Learning monotone Boolean functions.*
- *Learning $O(\log n) -$* CNF $\bigcap$ *$O(\log n) -$* DNF.

The ideas behind the above result use techniques from [1,3]. For a monotone Boolean function f, the standard closure algorithm uses both equivalence and membership queries to learn f using candidate functions g satisfying $\mathsf{g} \Rightarrow \mathsf{f}$. The need for membership can be removed using the following observation. Viewing $\neg\mathsf{f}$ as a monotone function on the inverted lattice, it is possible to learn f and $\neg\mathsf{f}$ simultaneously using candidate functions g,h, respectively, that satisfy $\mathsf{g} \Rightarrow \mathsf{h}$. The NP oracle is used to obtain an example a that either helps in learning f or in learning $\neg\mathsf{f}$; when no such example can be found, f was learned.

Theorem 6 *Any class* C *of Boolean functions that is exactly learnable using a polynomial number of membership queries (and unlimited computational power) is exactly learnable in expected polynomial time using a polynomial number of membership queries and access to an* NP *oracle.*

Moreover, any p-evaluatable class C *that is exactly learnable from a polynomially number membership queries (and unlimited computational power), is also learnable in deterministic polynomial time using a polynomial number of membership queries and access to a Σ_5^p oracle.*

Theorems 4 and 6 showed that information-theoretic learnability using equivalence and membership queries can be transformed into computational learnability at the expense of using the Σ_5^p and NP oracles, respectively.

Applications

The learning algorithm for Boolean circuits using equivalence queries and access to an NP oracle has found an application in complexity theory. Watanabe (see [9]) showed an improvement on a known theorem of Karp and Lipton [7]: if NP has polynomial-size circuits, then the polynomial-time hierarchy PH collapses to ZPP^{NP}.

Some techniques developed in Theorem 5 for exact learning using membership queries of monotone Boolean functions have found applications in data mining [5].

Open Problems

It is unknown if there are polynomial-time learning algorithms for Boolean circuits and DNF formulas using equivalence queries (without complexity-theoretic oracles). There are strong cryptographic evidence that Boolean circuits are not learnable in polynomial-time (see [2] and the references therein). The best running time for learning DNF formulas is $2^{\tilde{O}(n^{1/3})}$ as given by Klivans and Servedio [8]. It is unclear if membership queries help in this case.

Cross References

For related learning results, see ▶ Learning DNF Formulas and ▶ Learning Automata in this encyclopedia.

Recommended Reading

1. Angluin, D.: Queries and Concept Learning. Mach. Learn. **2**, 319–342 (1988)
2. Angluin, D., Kharitonov, M.: When Won't Membership Queries Help? J. Comput. Syst. Sci. **50**, 336–355 (1995)
3. Bshouty, N.H.: Exact Learning Boolean Function via the Monotone Theory. Inform. Comput. **123**, 146–153 (1995)
4. Bshouty, N.H., Cleve, R., Gavaldà, R., Kannan, S., Tamon, C.: Oracles and Queries That Are Sufficient for Exact Learning. J. Comput. Syst. Sci. **52**(3), 421–433 (1996)
5. Gunopolous, D., Khardon, R., Mannila, H., Saluja, S., Toivonen, H., Sharma, R.S.: Discovering All Most Specific Sentences. ACM Trans. Database Syst. **28**, 140–174 (2003)
6. Jerrum, M.R., Valiant, L.G., Vazirani, V.V.: Random Generation of Combinatorial Structures from a Uniform Distribution. Theor. Comput. Sci. **43**, 169–188 (1986)
7. Karp, R.M., Lipton, R.J.: Some Connections Between Nonuniform and Uniform Complexity Classes. In: Proc. 12th Ann. ACM Symposium on Theory of Computing, 1980, pp. 302–309
8. Klivans, A.R., Servedio, R.A.: Learning DNF in Time $2^{\tilde{O}(n^{1/3})}$. J. Comput. Syst. Sci. **68**, 303–318 (2004)
9. Köbler, J., Watanabe, O.: New Collapse Consequences of NP Having Small Circuits. SIAM J. Comput. **28**, 311–324 (1998)
10. Littlestone, N.: Learning Quickly When Irrelevant Attributes Abound: A New Linear-Threshold Algorithm. Mach. Learn. **2**, 285–318 (1987)
11. Sipser, M.: A complexity theoretic approach to randomness. In: Proc. 15th Annual ACM Symposium on Theory of Computing, 1983, pp. 330–334
12. Stockmeyer, L.J.: On approximation algorithms for #*P*. SIAM J. Comput. **14**, 849–861 (1985)

Learning Automata

2000; Beimel, Bergadano, Bshouty, Kushilevitz, Varricchio

AMOS BEIMEL[1], FRANCESCO BERGADANO[2], NADER H. BSHOUTY[3], EYAL KUSHILEVITZ[3], STEFANO VARRICCHIO[4]

[1] Ben-Gurion University, Beer Sheva, Israel
[2] University of Torino, Torino, Italy
[3] Technion, Haifa, Israel
[4] Department of Computer Science, University of Roma, Rome, Italy

Keywords and Synonyms

Computational learning; Machine learning; Multiplicity automata; Formal series; Boolean formulas; Multivariate polynomials

Problem Definition

This problem is concerned with the learnability of *multiplicity automata* in Angluin's *exact learning model* and applications to the learnability of functions represented by small multiplicity automata.

The Learning Model It is the *exact learning* model [2]: Let f be a *target* function. A learning algorithm may propose to an oracle, in each step, two kinds of queries: *membership queries* (MQ) and *equivalence queries* (EQ). In a MQ it may query for the value of the function f on a particular assignment z. The response to such a query is the value $f(z)$.[1] In a EQ it may propose to the oracle a hypothesis function h. If h is equivalent to f on all input assignments then the answer to the query is YES and the learning algorithm succeeds and halts. Otherwise, the answer to the equivalence query is NO and the algorithm receives a *counterexample*, i. e., an assignment z such that $f(z) \neq h(z)$. One says that the learner *learns* a class of functions C, if for every function $f \in C$ the learner outputs a hypothesis h that is equivalent to f and does so in time polynomial in the "size" of a shortest representation of f and the length of the longest counterexample. The exact learning model is strictly related to the *Probably Approximately Correct* (PAC) model of Valiant [19]. In fact, every equivalence query can be easily simulated by a sample of random examples. Therefore, learnability in the exact learning model also implies learnability in the PAC model with membership queries [2,19].

[1] If f is boolean this is the standard membership query.

Multiplicity Automata Let $\mathcal{K}$ be a field, Σ be an alphabet, ϵ be the empty string. A *multiplicity automaton* (MA) A of size r consists of $|\Sigma|$ matrices $\{\mu_\sigma : \sigma \in \Sigma\}$ each of which is an $r \times r$ matrix of elements from $\mathcal{K}$ and an r-tuple $\vec{\gamma} = (\gamma_1, \dots, \gamma_r) \in \mathcal{K}^r$. The automaton A defines a function $f_A : \Sigma^* \to \mathcal{K}$ as follows. First, define a mapping μ, which associates with every string in Σ^* an $r \times r$ matrix over $\mathcal{K}$, by $\mu(\epsilon) \triangleq \mathrm{ID}$, where ID denotes the *identity matrix*, and for a string $w = \sigma_1\sigma_2 \dots \sigma_n$, let $\mu(w) \triangleq \mu_{\sigma_1} \cdot \mu_{\sigma_2} \cdots \mu_{\sigma_n}$. A simple property of μ is that $\mu(x \circ y) = \mu(x) \cdot \mu(y)$, where $\circ$ denotes concatenation. Now, $f_A(w) \triangleq [\mu(w)]_1 \cdot \vec{\gamma}$ (where $[\mu(w)]_i$ denotes the ith row of the matrix $\mu(w)$). Let $f : \Sigma^* \to \mathcal{K}$ be a function. Associate with f an infinite matrix F, where each of its rows is indexed by a string $x \in \Sigma^*$ and each of its columns is indexed by a string $y \in \Sigma^*$. The (x, y) entry of F contains the value $f(x \circ y)$. The matrix F is called the *Hankel Matrix* of f. The xth row of F is denoted by F_x. The (x, y) entry of F may be therefore denoted as $F_x(y)$ and as $F_{x,y}$. The following result relates the size of the minimal MA for f to the rank of F (cf. [4] and references therein).

Theorem 1 *Let $f : \Sigma^* \to \mathcal{K}$ such that $f \not\equiv 0$ and let F be its Hankel matrix. Then, the size r of the smallest multiplicity automaton A such that $f_A \equiv f$ satisfies $r = \mathrm{rank}(F)$ (over the field $\mathcal{K}$).*

Key Results

The learnability of multiplicity automata has been proved in [7] and, independently, in [17]. In what follows let $\mathcal{K}$ be a field, $f : \Sigma^* \to \mathcal{K}$ be a function and F its Hankel matrix such that $r = \mathrm{rank}(F)$ (over $\mathcal{K}$).

Theorem 2 ([4]) *The function f is learnable by an algorithm in time $O(|\Sigma| \cdot r \cdot \mathrm{M}(r) + m \cdot r^3)$ using r equivalence queries and $O((|\Sigma| + \log m)r^2)$ membership queries, where m is the size of the longest counterexample obtained during the execution of the algorithm, and $\mathrm{M}(r)$ is the complexity of multiplying two $r \times r$ matrices.*

Some extensions of the above result can be found in [8,13,16]. In many cases of interest the domain of the target function f is not Σ^* but rather Σ^n for some value n, i. e., $f : \Sigma^n \to \mathcal{K}$. The length of counterexamples, in this case, is always n and so $m = n$. Denote by F^d the submatrix of F whose rows are strings in Σ^d and its columns are strings in Σ^{n-d} and let $r_{\max} = \max_{d=0}^{n} \mathrm{rank}(F^d)$ (where rank is taken over $\mathcal{K}$).

Theorem 3 ([4]) *The function f is learnable by an algorithm in time $O(|\Sigma| rn \cdot \mathrm{M}(r_{\max}))$ using $O(r)$ equivalence queries and $O((|\Sigma| + \log n) r \cdot r_{\max})$ membership queries.*

The time complexity of the two above results has been recently further improved [9].

Applications

The results of this section can be found in [3,4,5,6]. They show the learnability of various classes of functions as a consequence of Theorems 2 and 3. This can be done by proving that for every function f in the class in question, the corresponding Hankel matrix F has low rank. As is well known, any nondeterministic automaton can be regarded as a multiplicity automaton, whose associated function returns the number of accepting paths of the nondeterministic automaton on w. Therefore, the learnability of multiplicity automata gives a new algorithm for learning deterministic automata and unambiguous automata[2]. The learnability of deterministic automata has been proved in [1]. By [14], the class of deterministic automata contains the class of $O(\log n)$-term DNF, i. e., DNF formulae over n boolean variables with $O(\log n)$ number of terms. Hence, this class can be learned using multiplicity automata.

Classes of Polynomials

Theorem 4 *Let $p_{i,j} : \Sigma \to \mathcal{K}$ be arbitrary functions of a single variable ($1 \le i \le t$, $1 \le j \le n$). Let $g_i : \Sigma^n \to \mathcal{K}$ be defined by $\prod_{j=1}^{n} p_{i,j}(z_j)$. Finally, let $f : \Sigma^n \to \mathcal{K}$ be defined by $f = \sum_{i=1}^{t} g_i$. Let F be the Hankel matrix corresponding to f, and F^d the sub-matrices defined in the previous section. Then, for every $0 \le d \le n$, $\mathrm{rank}(F^d) \le t$.*

Corollary 5 *The class of functions that can be expressed as functions over $\mathrm{GF}(p)$ with t summands, where each summand T_i is a product of the form $p_{i,1}(x_1) \cdots p_{i,n}(x_n)$ (and $p_{i,j} : \mathrm{GF}(p) \to \mathrm{GF}(p)$ are arbitrary functions) is learnable in time $\mathrm{poly}(n, t, p)$.*

The above corollary implies as a special case the learnability of polynomials over $\mathrm{GF}(p)$. This extends the result of [18] from multi-linear polynomials to arbitrary polynomials. The algorithm of Theorem 3, for polynomials with n variables and t terms, uses $O(nt)$ equivalence queries and $O(t^2 n \log n)$ membership queries. The special case of the above class – the class of polynomials over $\mathrm{GF}(2)$ – was known to be learnable before [18]. Their algorithm uses $O(nt)$ equivalence queries and $O(t^3 n)$ membership queries. The following theorem extends the latter result to *infinite* fields, assuming that the functions $p_{i,j}$ are bounded-degree polynomials.

[2]A nondeterministic automata is *unambiguous* if for every $w \in \Sigma^*$ there is at most one accepting path.

Theorem 6 *The class of functions over a field $\mathcal{K}$ that can be expressed as t summands, where each summand T_i is of the form $p_{i,1}(x_1)\cdots p_{i,n}(x_n)$, and $p_{i,j} : \mathcal{K} \to \mathcal{K}$ are univariate polynomials of degree at most k, is learnable in time* $\text{poly}(n, t, k)$. *Furthermore, if $|\mathcal{K}| \geq nk + 1$ then this class is learnable from membership queries only in time* $\text{poly}(n, t, k)$ *(with small probability of error).*

Classes of Boxes

Let $[\ell]$ denotes the set $\{0, 1, \ldots, \ell - 1\}$. A box in $[\ell]^n$ is defined by two corners $(a_1, \ldots, a_n)$ and $(b_1, \ldots, b_n)$ (in $[\ell]^n$) as follows:

$$B_{a_1,\ldots,a_n,b_1,\ldots,b_n} = \{(x_1, \ldots, x_n) \colon \forall i,\ a_i \leq x_i \leq b_i\} .$$

A box can be represented by its characteristic function in $[\ell]^n$. The following result concerns a more general class of functions.

Theorem 7 *Let $p_{i,j} : \Sigma \to \{0, 1\}$ be arbitrary functions of a single variable ($1 \leq i \leq t$, $1 \leq j \leq n$). Let $g_i : \Sigma^n \to \{0, 1\}$ be defined by $\prod_{j=1}^n p_{i,j}(z_j)$. Assume that there is no point $x \in \Sigma^n$ such that $g_i(x) = 1$ for more than s functions g_i. Finally, let $f : \Sigma^n \to \{0, 1\}$ be defined by $f = \bigvee_{i=1}^t g_i$. Let F be the Hankel matrix corresponding to f. Then, for every field $\mathcal{K}$ and for every $0 \leq d \leq n$,* $\text{rank}(F^d) \leq \sum_{i=1}^s \binom{t}{i}$.

Corollary 8 *The class of unions of disjoint boxes can be learned in time* $\text{poly}(n, t, \ell)$ *(where t is the number of boxes in the target function). The class of unions of* $O(\log n)$ *boxes can be learned in time* $\text{poly}(n, \ell)$.

Classes of DNF Formulae

The learnability of DNF formulae has been widely investigated. The following special case of Corollary 5 solves an open problem of [18]:

Corollary 9 *The class of functions that can be expressed as exclusive-OR of t (not necessarily monotone) monomials is learnable in time* $\text{poly}(n, t)$.

While Corollary 9 does not refer to a subclass of DNF, it already implies the learnability of disjoint (i. e., satisfy-1) DNF. Since DNF is a special case of union of boxes (with $\ell = 2$), one obtains also the learnability of disjoint DNF from Corollary 8. Positive results for satisfy-*s* DNF (i. e., DNF formulae in which each assignment satisfies at most *s* terms) can be obtained, with larger values of *s*. The following two important corollaries follow from Theorem 7. Note that Theorem 7 holds in any field. For convenience (and efficiency), let $\mathcal{K} = \text{GF}(2)$.

Theorem 10 *Let $f = T_1 \vee T_2 \vee \cdots \vee T_t$ be a satisfy-s DNF (that is, each T_i is a monomial). Let F be the Hankel matrix corresponding to f. Then,* $\text{rank}(F^d) \leq \sum_{i=1}^s \binom{t}{i} \leq t^s$.

Corollary 11 *The class of satisfy-s DNF formulae, for* $s = O(1)$, *is learnable in time* $\text{poly}(n, t)$.

Corollary 12 *The class of satisfy-s, t-term DNF formulae is learnable in time* $\text{poly}(n)$ *for the following choices of s and t: (1)* $t = O(\log n)$*; (2)* $t = \text{polylog}(n)$ *and* $s = O(\log n/\log\log n)$*; (3)* $t = 2^{O(\log n/\log\log n)}$ *and* $s = O(\log\log n)$.

Classes of Decision Trees

The algorithm of Theorem 3 efficiently learns the class of disjoint DNF formulae. This includes the class of decision trees. Therefore, decision trees of size t on n variables are learnable using $O(tn)$ equivalence queries and $O(t^2 n \log n)$ membership queries. This is better than the best known algorithm for decision trees [11] (which uses $O(t^2)$ equivalence queries and $O(t^2 n^2)$ membership queries). The following results concern more general classes of decision trees.

Corollary 13 *Consider the class of decision trees that compute functions $f : \text{GF}(p)^n \to \text{GF}(p)$ as follows: each node v contains a query of the form "$x_i \in S_v$?", for some $S_v \subseteq \text{GF}(p)$. If $x_i \in S_v$ then the computation proceeds to the left child of v and if $x_i \notin S_v$ the computation proceeds to the right child. Each leaf ℓ of the tree is marked by a value $\gamma_\ell \in \text{GF}(p)$ which is the output on all the assignments which reach this leaf. Then, this class is learnable in time* $\text{poly}(n, |L|, p)$*, where L is the set of leaves.*

The above result implies the learnability of decision trees with "greater-than" queries in the nodes, solving a problem of [11]. Every decision tree with "greater-than" queries that computes a boolean function can be expressed as the union of disjoint boxes. Hence, this case can also be derived from Corollary 8. The next theorem will be used to learn more classes of decision trees.

Theorem 14 *Let $g_i : \Sigma^n \to \mathcal{K}$ be arbitrary functions ($1 \leq i \leq \ell$). Let $f : \Sigma^n \to \mathcal{K}$ be defined by $f = \prod_{i=1}^\ell g_i$. Let F be the Hankel matrix corresponding to f, and G_i be the Hankel matrix corresponding to g_i. Then,* $\text{rank}(F^d) \leq \prod_{i=1}^\ell \text{rank}(G_i^d)$.

This theorem has some interesting applications. The first application states that arithmetic circuits of depth two with multiplication gate of fan-in $O(\log n)$ at the top level and addition gates with unbounded fan-in in the bottom level are learnable.

Corollary 15 *Let C be the class of functions that can be expressed in the following way: Let $p_{i,j} : \Sigma \to \mathcal{K}$ be arbitrary functions of a single variable ($1 \le i \le \ell$, $1 \le j \le n$). Let $\ell = O(\log n)$ and $g_i : \Sigma^n \to \mathcal{K}$ ($1 \le i \le \ell$) be defined by $\Sigma_{j=1}^{n} p_{i,j}(z_j)$. Finally, let $f : \Sigma^n \to \mathcal{K}$ be defined by $f = \prod_{i=1}^{\ell} g_i$. Then, C is learnable in time* $\mathrm{poly}(n, |\Sigma|)$.

Corollary 16 *Consider the class of decision trees of depth s, where the query at each node v is a boolean function f_v with $r_{\max} \le t$ (as defined in Section "Key Results") such that $(t+1)^s = \mathrm{poly}(n)$. Then, this class is learnable in time* $\mathrm{poly}(n, |\Sigma|)$.

The above class contains, for example, all the decision trees of depth $O(\log n)$ that contain in each node a term or a XOR of a subset of variables as defined in [15] (in this case $r_{\max} \le 2$).

Negative Results

In [4] some limitation of the learnability via the automaton representation has been proved. One can show that the main algorithm does not efficiently learn several important classes of functions. More precisely, these classes contain functions f that have no "small" automaton, i. e., by Theorem 1, the corresponding Hankel matrix F is "large" over every field $\mathcal{K}$.

Theorem 17 *The following classes are not learnable in time polynomial in n and the formula size using multiplicity automata (over any field $\mathcal{K}$): DNF, Monotone DNF, 2-DNF, Read-once DNF, k-term DNF, for $k = \omega(\log n)$, Satisfy-s DNF, for $s = \omega(1)$, Read-j satisfy-s DNF, for $j = \omega(1)$ and $s = \Omega(\log n)$.*

Some of these classes are known to be learnable by other methods, some are natural generalizations of classes known to be learnable as automata ($O(\log n)$-term DNF [11,12,14], and satisfy-s DNF for $s = O(1)$ (Corollary 11)) or by other methods (read-j satisfy-s for $js = O(\log n / \log\log n)$ [10]), and the learnability of some of the others is still an open problem.

Cross References

► Learning Constant-Depth Circuits
► Learning DNF Formulas

Recommended Reading

1. Angluin, D.: Learning regular sets from queries and counterexamples. Inf. Comput. **75**, 87–106 (1987)
2. Angluin, D.: Queries and concept learning. Mach. Learn. **2**(4), 319–342 (1988)
3. Beimel, A., Bergadano, F., Bshouty, N.H., Kushilevitz, E., Varricchio, S.: On the applications of multiplicity automata in learning. In: Proc. of the 37th Annu. IEEE Symp. on Foundations of Computer Science, pp. 349–358, IEEE Comput. Soc. Press, Los Alamitos (1996)
4. Beimel, A., Bergadano, F., Bshouty, N.H., Kushilevitz, E., Varricchio, S.: Learning Functions Represented as Multiplicity Automata. J. ACM **47**, 506–530 (2000)
5. Beimel, A., Kushilevitz, E.: Learning boxes in high dimension. In: Ben-David S. (ed.) 3rd European Conf. on Computational Learning Theory (EuroCOLT '97), Lecture Notes in Artificial Intelligence, vol. 1208, pp. 3–15. Springer, Berlin (1997) Journal version: Algorithmica **22**, 76–90 (1998)
6. Bergadano, F., Catalano, D., Varricchio, S.: Learning sat-*k*-DNF formulas from membership queries. In: Proc. of the 28th Annu. ACM Symp. on the Theory of Computing, pp. 126–130. ACM Press, New York (1996)
7. Bergadano, F., Varricchio, S.: Learning behaviors of automata from multiplicity and equivalence queries. In: Proc. of 2nd Italian Conf. on Algorithms and Complexity. Lecture Notes in Computer Science, vol. 778, pp. 54–62. Springer, Berlin (1994). Journal version: SIAM J. Comput. **25**(6), 1268–1280 (1996)
8. Bergadano, F., Varricchio, S.: Learning behaviors of automata from shortest counterexamples. In: EuroCOLT '95, Lecture Notes in Artificial Intelligence, vol. 904, pp. 380–391. Springer, Berlin (1996)
9. Bisht, L., Bshouty, N.H., Mazzawi, H.: On Optimal Learning Algorithms for Multiplicity Automata. In: Proc. of 19th Annu. ACM Conf. Comput. Learning Theory, Lecture Notes in Computer Science. vol. 4005, pp. 184–198. Springer, Berlin (2006)
10. Blum, A., Khardon, R., Kushilevitz, E., Pitt, L., Roth, D.: On learning read-*k*-satisfy-*j* DNF. In: Proc. of 7th Annu. ACM Conf. on Comput. Learning Theory, pp. 110–117. ACM Press, New York (1994)
11. Bshouty, N.H.: Exact learning via the monotone theory. In: Proc. of the 34th Annu. IEEE Symp. on Foundations of Computer Science, pp. 302–311. IEEE Comput. Soc. Press, Los Alamitos (1993). Journal version: Inform. Comput. **123**(1), 146–153 (1995)
12. Bshouty, N.H.: Simple learning algorithms using divide and conquer. In: Proc. of 8th Annu. ACM Conf. on Comput. Learning Theory, pp. 447–453. ACM Press, New York (1995). Journal version: Computational Complexity, **6**, 174–194 (1997)
13. Bshouty, N.H., Tamon, C., Wilson, D.K.: Learning Matrix Functions over Rings. Algorithmica **22**(1/2), 91–111 (1998)
14. Kushilevitz, E.: A simple algorithm for learning $O(\log n)$-term DNF. In: Proc. of 9th Annu. ACM Conf. on Comput. Learning Theory, pp 266–269, ACM Press, New York (1996). Journal version: Inform. Process. Lett. **61**(6), 289–292 (1997)
15. Kushilevitz, E., Mansour, Y.: Learning decision trees using the Fourier spectrum. SIAM J. Comput. **22**(6), 1331–1348 (1993)
16. Melideo, G., Varricchio, S.: Learning unary output two-tape automata from multiplicity and equivalence queries. In: ALT '98. Lecture Notes in Computer Science, vol. 1501, pp. 87–102. Springer, Berlin (1998)
17. Ohnishi, H., Seki, H., Kasami, T.: A polynomial time learning algorithm for recognizable series. IEICE Transactions on Information and Systems, **E77-D(10)**(5), 1077–1085 (1994)
18. Schapire, R.E., Sellie, L.M.: Learning sparse multivariate polynomials over a field with queries and counterexamples. J. Comput. Syst. Sci. **52**(2), 201–213 (1996)
19. Valiant, L.G.: A theory of the learnable. Commun. ACM **27**(11), 1134–1142 (1984)

Learning Constant-Depth Circuits

1993; Linial, Mansour, Nisan

Rocco Servedio
Department of Computer Science, Columbia University, New York, NY, USA

Keywords and Synonyms

Learning AC^0 circuits

Problem Definition

This problem deals with learning "simple" Boolean functions $f : \{0,1\}^n \rightarrow \{-1,1\}$ from uniform random labeled examples. In the basic uniform distribution PAC framework, the learning algorithm is given access to a *uniform random example oracle* $EX(f, U)$ which, when queried, provides a labeled random example $(x, f(x))$ where x is drawn from the uniform distribution U over the Boolean cube $\{0,1\}^n$. Successive calls to the $EX(f, U)$ oracle yield independent uniform random examples. The goal of the learning algorithm is to output a representation of a hypothesis function $h : \{0,1\}^n \rightarrow \{-1,1\}$ which with high probability has high accuracy; formally, for any $\epsilon, \delta > 0$, given ϵ and δ the learning algorithm should output an h which with probability at least $1-\delta$ has $\Pr_{x\in U}[h(x) \neq f(x)] \leq \epsilon$.

Many variants of the basic framework described above have been considered. In the *distribution-independent* PAC learning model, the random example oracle is $EX(f, \mathcal{D})$ where $\mathcal{D}$ is an arbitrary (and unknown to the learner) distribution over $\{0,1\}^n$; the hypothesis h should now have high accuracy with respect to $\mathcal{D}$, i. e. with probability $1-\delta$ it must satisfy $\Pr_{x\in\mathcal{D}}[h(x) \neq f(x)] \leq \epsilon$. Another variant that has been considered is when the distribution $\mathcal{D}$ is assumed to be an unknown *product distribution*; such a distribution is defined by n parameters $0 \leq p_1, \ldots, p_n \leq 1$, and a draw from $\mathcal{D}$ is obtained by independently setting each bit x_i to 1 with probability p_i. Yet another variant is to consider learning with the help of a *membership oracle*: this is a "black-box" oracle $MQ(f)$ for f which, when queried on an input $x \in \{0,1\}^n$, returns the value of $f(x)$. The model of uniform distribution learning with a membership oracle has been well studied, see e. g. [4,11].

There are many ways to make precise the notion of a "simple" Boolean function; one common approach is to stipulate that the function be computed by a Boolean circuit of some restricted form. A circuit of *size s and depth d* consists of s AND and OR gates (of unbounded fanin) in which the longest path from any input literal $x_1, \ldots, x_n, \overline{x}_1, \ldots, \overline{x}_n$ to the output node is of length d. Note that a circuit of size s and depth 2 is simply a CNF formula or DNF formula. The complexity class consisting of those Boolean functions computed by poly(n)-size, $O(1)$-depth circuits is known as *nonuniform* AC^0.

Key Results

Positive Results

Linial et al. [12] showed that almost all of the "Fourier weight" of any constant-depth circuit is on low-degree Fourier coefficients:

Lemma 1 *Let $f : \{0,1\}^n \rightarrow \{-1,1\}$ be a Boolean function that is computed by a circuit of size s and depth d. Then for any integer $t \geq 0$,*

$$\sum_{S\subset\{1,\ldots,n\},|S|>t} \hat{f}(S)^2 \leq 2s2^{-t^{1/d}/20}.$$

(Hastad [3] has given a refined version of Lemma 1 with slightly sharper bounds; see also [17] for a streamlined proof.) They also showed that any Boolean function can be well approximated by approximating its Fourier spectrum:

Lemma 2 *Let $f : \{0,1\}^n \rightarrow \{-1,1\}$ be any Boolean function and let $g : \{0,1\}^n \rightarrow \boldsymbol{R}$ be an arbitrary function such that $\sum_{S\subseteq\{1,\ldots,n\}}(\hat{f}(S) - \hat{g}(S))^2 \leq \epsilon$. Then $\Pr_{x\in U}[f(x) \neq sign(g(x))] \leq \epsilon$.*

Using the above two results together with a procedure that estimates all the "low-order" Fourier coefficients, they obtained a quasipolynomial-time algorithm for learning constant-depth circuits:

Theorem 3 *There is an $n^{(O(\log(n/\epsilon)))^d}$-time algorithm that learns any poly(n)-size, depth-d Boolean circuit to accuracy ϵ with respect to the uniform distribution, using uniform random examples only.*

Furst et al. [2] extended this result to learning under constant-bounded product distributions. A product distribution $\mathcal{D}$ is said to be *constant-bounded* if each of its n parameters $p_1, \cdots, p_n$ is bounded away from 0 and 1, i. e. satisfies $\min\{p_i, 1-p_i\} = \Theta(1)$.

Theorem 4 *There is an $n^{(O(\log(n/\epsilon)))^d}$-time algorithm that learns any poly(n)-size, depth-d Boolean circuit to accuracy ϵ given random examples drawn from any constant-bounded product distribution.*

By combining the Fourier arguments of Linial et al. with hypothesis boosting, Jackson et al. [5] were able to extend

Theorem 3 to a broader class of circuits, namely constant-depth AND/OR circuits that additionally contain (a limited number of) *majority gates*. A majority gate over r Boolean inputs is a binary gate which outputs "true" if and only if at least half of its r Boolean inputs are set to "true".

Theorem 5 *There is an $n^{\log^{O(1)}(n/\epsilon)}$-time algorithm that learns any poly(n)-size, constant-depth Boolean circuit that contains polylog(n) many majority gates to accuracy ϵ with respect to the uniform distribution, using uniform random examples only.*

Negative Results

Kharitonov [7] showed that under a strong but plausible cryptographic assumption, the algorithmic result of Theorem 3 is essentially optimal. A *Blum integer* is an integer $N = P \cdot Q$ where both P and Q are congruent to 3 modulo 4. Kharitonov proved that if the problem of factoring a randomly chosen n-bit Blum integer is 2^{n^ϵ}-hard for some fixed $\epsilon > 0$, then any algorithm that (even weakly) learns polynomial-size depth-d circuits must run in time $2^{\log^{\Omega(d)} n}$, even if it is only required to learn under the uniform distribution and can use a membership oracle. This implies that there is no polynomial-time algorithm for learning polynomial-size, depth-d circuits (for d larger than some absolute constant).

Using a cryptographic construction of Naor and Reingold [14], Jackson et al. [5] proved a related result for circuits with majority gates. They showed that under Kharitonov's assumption, any algorithm that (even weakly) learns depth-5 circuits consisting of $\log^k n$ many majority gates must run in time $2^{\log^{\Omega(k)} n}$ time, even if it is only required to learn under the uniform distribution and can use a membership oracle.

Applications

The technique of learning by approximating most of the Fourier spectrum (Lemma 2 above) has found many applications in subsequent work on uniform distribution learning. It is a crucial ingredient in the current state-of-the-art algorithms for learning monotone DNF formulas [16], monotone decision trees [15], and intersections of halfspaces [8] from uniform random examples only. Combined with a membership-oracle based procedure for identifying large Fourier coefficients, this technique is at the heart of an algorithm for learning decision trees [11]; this algorithm in turn plays a crucial role in the celebrated polynomial-time algorithm of Jackson [4] for learning polynomial-size depth-2 circuits under the uniform distribution.

The ideas of Linial et al. have also been applied for the difficult problem of *agnostic learning*. In the agnostic learning framework there is a joint distribution $\mathcal{D}$ over example-label pairs $\{0,1\}^n \times \{-1,1\}$; the goal of an agnostic learning algorithm for a class C of functions is to construct a hypothesis h such that $\Pr_{(x,y)\in\mathcal{D}}[h(x) \neq y] \leq \min_{f\in C} \Pr_{(x,y)\in\mathcal{D}}[f(x) \neq y] + \epsilon$. Kalai et al. [6] gave agnostic learning algorithms for halfspaces and related classes via an algorithm which may be viewed as a generalization of Linial et al.'s algorithm to a broader class of distributions.

Finally, there has been some applied work on learning using Fourier representations as well [13].

Open Problems

Perhaps the most outstanding open question related to this work is whether polynomial-size circuits of depth two – i. e. DNF formulas – can be learned in polynomial time from uniform random examples only. Blum [1] has offered a cash prize for a solution to a restricted version of this problem. A hardness result for learning DNF would also be of great interest; recent work of Klivans and Sherstov [10] gives a hardness result for learning ANDs of majority gates, but hardness for DNF (ANDs of ORs) remains an open question.

Another open question is whether the quasipolynomial-time algorithms for learning constant-depth circuits under uniform distributions and product distributions can be extended to the general distribution-independent model. Known results in complexity theory imply that quasipolynomial-time distribution-independent learning algorithms for constant-depth circuits would follow from the existence of efficient linear threshold learning algorithms with a sufficiently high level of tolerance to "malicious" noise. Currently no nontrivial distribution-independent algorithms are known for learning circuits of depth 3; for depth-2 circuits the best known running time in the distribution-independent setting is the $2^{\tilde{O}(n^{1/3})}$-time algorithm of Klivans and Servedio [9].

A third direction for future work is to extend the results of [5] to a broader class of circuits. Can constant-depth circuits augmented with MOD_p gates, or with *weighted* majority gates, be learned in quasipolynomial time? [5] discusses the limitations of current techniques to address these extensions.

Cross References

▶ Cryptographic Hardness of Learning
▶ Learning DNF Formulas
▶ PAC Learning
▶ Statistical Query Learning

Recommended Reading

1. Blum, A.: Learning a function of r relevant variables (open problem). In: Proceedings of the 16th Annual Conference on Learning Theory, pp. 731–733, Washington, 24–27 August 2003
2. Furst, M., Jackson, J., Smith, S.: Improved learning of AC^0 functions. In: Proceedings of the Fourth Annual Workshop on Computational Learning Theory, pp. 317–325, Santa Cruz, (1991)
3. Håstad, J.: A slight sharpening of LMN. J. Comput. Syst. Sci. **63**(3), 498–508 (2001)
4. Jackson, J.: An efficient membership-query algorithm for learning DNF with respect to the uniform distribution. J. Comput. Syst. Sci. **55**, 414–440 (1997)
5. Jackson, J., Klivans, A., Servedio, R.: Learnability beyond AC^0. In: Proceedings of the 34th ACM Symposium on Theory of Computing, pp. 776–784, Montréal, 23–25 May 2002
6. Kalai, A., Klivans, A., Mansour, Y., Servedio, R.: Agnostically learning halfspaces. In: Proceedings of the 46th IEEE Symposium on Foundations of Computer Science (FOCS), pp. 11–20, Pittsburgh, PA, USA, 23–25 October 2005
7. Kharitonov, M.: Cryptographic hardness of distribution-specific learning. In: Proceedings of the 25th Annual Symposium on Theory of Computing, pp. 372–381. (1993)
8. Klivans, A., O'Donnell, R., Servedio, R.: Learning intersections and thresholds of halfspaces. J. Comput. Syst. Sci. **68**(4), 808–840 (2004)
9. Klivans, A., Servedio, R.: Learning DNF in time $2^{\tilde{O}(n^{1/3})}$. J. Comput. Syst. Sci. **68**(2), 303–318 (2004)
10. Klivans, A., Sherstov, A.: Cryptographic hardness results for learning intersections of halfspaces. In: Proceedings of the 47th Annual Symposium on Foundations of Computer Science, pp. 553–562, Berkeley, 22–24 October 2006
11. Kushilevitz, E., Mansour, Y.: Learning decision trees using the Fourier spectrum. SIAM J. Comput. **22**(6), 1331–1348 (1993)
12. Linial, N., Mansour, Y., Nisan, N.: Constant depth circuits, Fourier transform and learnability. J. ACM **40**(3), 607–620 (1993)
13. Mansour, Y., Sahar, S.: Implementation Issues in the Fourier Transform Algorithm. Mach. Learn. **40**(1), 5–33 (2000)
14. Naor, M., Reingold, O.: Number-theoretic constructions of efficient pseudo-random functions. J. ACM **51**(2), 231–262 (2004)
15. O'Donnell, R., Servedio, R.: Learning monotone decision trees in polynomial time. In: Proceedings of the 21st Conference on Computational Complexity (CCC), pp. 213–225, Prague, 16–20 July 2006
16. Servedio, R.: On learning monotone DNF under product distributions. Inform Comput **193**(1), 57–74 (2004)
17. Stefankovic, D.: Fourier transforms in computer science. Masters thesis, TR-2002-03, University of Chicago (2002)

Learning DNF Formulas

1997; Jackson

JEFFREY C. JACKSON
Department of Mathematics and Computer Science, Duquesne University, Pittsburgh, PA, USA

Keywords and Synonyms

Sum of products notation; Learning disjunctive normal form formulas (or expressions); Learning sums of products

Problem Definition

A Disjunctive Normal Form (DNF) expression is a Boolean expression written as a disjunction of *terms*, where each term is the conjunction of Boolean variables that may or may not be negated. For example, $(v_1 \wedge \overline{v_2}) \vee (v_2 \wedge v_3)$ is a two-term DNF expression over three variables. DNF expressions occur frequently in digital circuit design, where DNF is often referred to as sum of products notation. From a learning perspective, DNF expressions are of interest because they provide a natural representation for certain types of expert knowledge. For example, the conditions under which complex tax rules apply can often be readily represented as DNFs. Another nice property of DNF expressions is their universality: every n-bit Boolean function $f\colon \{0,1\}^n \to \{0,1\}$ can be represented as a DNF expression F over at most n variables.

Informally, the problem to be addressed is the following. A learning algorithm is given access to an oracle $MEM(f)$ that, for some fixed integer $n > 0$, on an n-bit input will return a 1-bit response. The output of the oracle is determined by an n-bit Boolean function f that can be represented by an s-term DNF expression F over n variables. All that is known about f and F is n. The algorithm's goal is to produce, with high probability over the random choices it makes, an n-bit Boolean function h that agrees with f on all but a small fraction of the 2^n elements of the domain of f and h. Furthermore, the algorithm must run in time polynomial in n and s (and other parameters given in the formal problem definition). The algorithm is not required to output a DNF representation of the approximating function h, but h must be computable in time comparable to that needed to evaluate f. In particular, $h(x)$ should be computable in time polynomial in n and s for all $x \in \{0,1\}^n$.

In the following formal problem definition, U_n represents the uniform distribution over $\{0,1\}^n$.

Problem 1 (UDNFL)
Input: *Positive integer n; $\varepsilon, \delta > 0$; oracle $MEM(f)$ for $f : \{0,1\}^n \to \{0,1\}$ expressible as DNF with s terms over n variables.*
Output: *With probability at least $1-\delta$ over the random choices made by the algorithm, a function $h\colon \{0,1\}^n \to \{0,1\}$ (not necessarily a DNF expression) such that*

$\Pr_{x\sim U_n}[h(x) \neq f(x)] < \varepsilon$. *The algorithm must run in time polynomial in n, s,* $1/\varepsilon$, *and* $1/\delta$, *and for all* $x \in \{0,1\}^n$, *h(x) must be computable in time polynomial in n and s.*

Threshold of Parities (TOP) is another interesting universal representation for Boolean functions. For a and x in $\{0,1\}^n$, the *even parity function* $e_a(x)$ returns 1 if the dot product $a \cdot x$ is even and 0 otherwise. That is, the output is 1 if the parity of the bits in x indexed by a is even and 0 otherwise. Similarly, define the *odd parity function* $o_a(x)$ to return 1 if the parity of the bits in x indexed by a is odd and 0 otherwise. A *parity function* is either an even or an odd parity function. Then a *TOP representation of size s* is defined by a collection of s parity functions $(p_1, p_2, \ldots, p_s)$, where p_i is allowed to be the same as p_j for $i \neq j$ (i. e., there may be fewer than s distinct functions in the collection). The value of a TOP F on input x is the majority value of $p_i(x)$ over the s parity functions defining F (with value 0 in the case of no majority).

Problem 2 (UTOPL)
Input: *Positive integer n;* $\varepsilon, \delta > 0$; *oracle MEM(f) for* $f : \{0,1\}^n \to \{0,1\}$ *expressible as TOP of size s over n variables.*
Output: *With probability at least* $1-\delta$ *over the random choices made by the algorithm, a function* $h : \{0,1\}^n \to \{0,1\}$ *(not necessarily a TOP) such that* $\Pr_{x\sim U_n}[h(x) \neq f(x)] < \varepsilon$. *The algorithm must run in time polynomial in n, s,* $1/\varepsilon$, *and* $1/\delta$, *and for all* $x \in \{0,1\}^n$, *h(x) must be computable in time polynomial in n and s.*

TOP and DNF representations of the same Boolean function f can be related as follows. For every $f : \{0,1\}^n \to \{0,1\}$, if f can be represented by an s-term DNF expression then there is a TOP representation of f of size $O(ns^2)$ [7]. On the other hand, a DNF of size 2^{n-1} is required to represent the parity function $e_{1^n}()$ (even parity in which all bits are relevant). So the DNF expression for a function is at most polynomially more succinct than the optimal equivalent TOP expression, whereas TOP expressions may be exponentially more succinct than the optimal equivalent DNF expressions.

From a learning viewpoint, this means that UTOPL is a harder problem then UDNFL. This is because the only difference between the problems is in how the size parameter s is defined, with larger values of s allowing the learning algorithm more time. Thus, since the DNF size of a function f is never much smaller than the TOP size of f and may be much larger, the learning algorithm is effectively allowed more time for DNF learning than it is for TOP learning.

Another direction in which the DNF problem can naturally be extended is to non-Boolean inputs. A DNF with s terms can be viewed as a union of s subcubes of the Boolean hypercube $\{0,1\}^n$, with the portion of the hypercube covered by this union corresponding to the 1 values of the DNF. Similarly, for any fixed positive integer b, a function $f : \{0,1,\ldots,b-1\}^n \to \{0,1\}$ can be defined as a union of rectangles over $\{0,1,\ldots,b-1\}^n$, where a *rectangle* is the set of all elements in a Cartesian product $\prod_{i=1}^{n}\{\ell_i, \ell_i+1, \ldots, u_i\}$, where for all i, $0 \le \ell_i \le u_i < b$. As in the Boolean case, the 1 values of f correspond exactly to those inputs included in this union of rectangles. Such a representation of f as a union of s rectangles will be called a *UBOX of size s*. Defining U_n^b to be the uniform distribution over $\{0,1,\ldots,b-1\}^n$, this gives rise to the following problem:

Problem 3 (UUBOXL) Input: *Positive integers n and b;* $\varepsilon, \delta > 0$; *oracle MEM(f) for* $f : \{0,1,\ldots,b-1\}^n \to \{0,1\}$ *expressible as UBOX of size s over n variables.*

Output: *With probability at least* $1-\delta$ *over the random choices made by the algorithm, a function* $h : \{0,1,\ldots,b-1\}^n \to \{0,1\}$ *(not necessarily a UBOX) such that* $\Pr_{x\sim U_n^b}[h(x) \neq f(x)] < \varepsilon$. *The algorithm must run in time polynomial in n, s,* $1/\varepsilon$, *and* $1/\delta$, *and for all* $x \in \{0,1\}^n$, *h(x) must be computable in time polynomial in n and s (b is taken to be a constant).*

Key Results

Theorem 1 *There is an algorithm—the Harmonic Sieve[7,9,10]—that solves both* UDNFL *and* UTOPL.

The run time[1] of the original version of the Harmonic Sieve [7] is $\tilde{O}(ns^{10}/\varepsilon^{12+c})$, where c is an arbitrarily small positive constant and the $\tilde{O}()$ notation is the same as big-O notation except that logarithmic factors are suppressed (in particular, the run-time dependence on $1/\delta$ is logarithmic). This bound was improved in [4] and [11] to $\tilde{O}(ns^6/\varepsilon^2)$, and Feldman [6] has further improved the run time to $\tilde{O}(ns^4/\varepsilon)$. All of the improvements use the same overall algorithmic structure as the Harmonic Sieve, but some components of the structure are replaced with more efficient approaches. The output h of the original Sieve is a TOP, but this is not the case for the more efficient versions of the algorithm.

Learnability of TOP implies learnability of several other classes that are, for purposes of uniform learning, special cases of TOP [7]. This includes the class of those functions that can be defined as a majority of arbitrary $(\log n)$-bit Boolean functions (size measure is the number

[1]See [4] for an explanation of an error in the time bound given in [7]

of functions) and the class of those functions that can be expressed as a *parity-DNF*, that is, as an OR of ANDs of parity functions (size measure is the number of ANDs).

Theorem 2 *A variation of the Harmonic Sieve solves* UUBOXL.

An algorithm for UUBOXL is given in [7].

Applications

An extended version of the Harmonic Sieve can tolerate false responses by the oracle *MEM*(*f*). In particular, in the *uniform persistent classification noise* learning model, a constant noise rate $0 \leq \eta < 1/2$ is fixed and an oracle $MEM^{\eta}(f)$ is defined as follows: if the query x has not been presented to the oracle previously, then $MEM^{\eta}(f)$ returns $f(x)$ with probability $1-\eta$ and the complement $\overline{f(x)}$ with probability η. If x has been presented to the oracle previously, then it returns the same response that it did on the first query with x. Jackson et al. [8] showed how to modify the Harmonic Sieve to efficiently learn DNF (and TOP, although the referenced paper does not state this) in the uniform persistent classification noise model.

Bshouty and Jackson [3] defined a *uniform quantum example oracle* $QEX(f, U_n)$ that, for a fixed unknown function $f : \{0,1\}^n \rightarrow \{0,1\}$, produces a quantum superposition of the 2^n labeled-example pairs $\langle x, f(x)\rangle$, one pair for each $x \in \{0,1\}^n$. All pairs have the same amplitude, $2^{-n/2}$. Bshouty and Jackson then showed that such an oracle $QEX(f, U_n)$ cannot simulate the oracle *MEM*(*f*) used by the Harmonic Sieve to learn DNF and TOP, and therefore is a weaker form of oracle. Nevertheless, building on the Harmonic Sieve, they gave an efficient quantum algorithm for learning DNF and TOP from a uniform quantum example oracle.

Bshouty et al. [5] defined a model of learning from uniform random walks over $\{0,1\}^n$. Unlike the oracle *MEM*(*f*), where the learning algorithm actively selects examples to be queried, in the random walk model the learner passively accepts random examples from the random walk oracle. Building in part on the Harmonic Sieve, Bshouty et al. showed that DNF is efficiently learnable in the uniform random walk model. In addition, for fixed $0 \leq \rho \leq 1$, Bshouty et al. defined a *ρ-Noise Sensitivity example oracle* $NS - EX_{\rho}(f)$ that, when invoked, selects an input $x \in \{0,1\}^n$ uniformly at random, forms an input y by flipping each bit of x independently at random with probability $\frac{1}{2}(1-\rho)$, and returns the quadruple $\langle x, f(x), y, f(y)\rangle$. This oracle is shown to be no more powerful than the uniform random walk oracle, and an algorithm based in part on the Sieve is presented that, for any constant $\rho \in [0,1]$, efficiently learns DNF using $NS - EX_{\rho}(f)$. However, the question of whether or not TOP is efficiently learnable in either of these models is left open.

Atici and Servedio [1] have given a generalized version of the Harmonic Sieve that can, among other things, learn an interesting subset of the class of unions of rectangles over $\{0,1,\ldots,b-1\}^n$ for non-constant b.

Open Problems

A key open problem involves relaxing the power of the oracle used. For instance, given a function $f\colon \{0,1\}^n \rightarrow \{0,1\}$, a *uniform example oracle for* f $EX(f, U_n)$ is an oracle that, on every query, randomly selects an input x according to U_n and returns the value $f(x)$. If the definition of UDNFL is changed so that $EX(f, U_n)$ is provided rather than *MEM*(*f*), is UDNFL still solvable? There is at least some reason to believe that the answer is no (see, e. g., [2]). The apparently simpler question of whether or not the class of *monotone DNF expressions* (DNF expressions with no negated variables) is efficiently uniform learnable from an example oracle is also still open, and there is less reason to doubt that an algorithm solving this problem will be discovered.

Cross References

► Learning Constant-Depth Circuits
► Learning Heavy Fourier Coefficients of Boolean Functions
► PAC Learning

Recommended Reading

1. Atici, A., Servedio, R.A.: Learning unions of $\omega(1)$-dimensional rectangles. In: Proceedings of 17th Algorithmic Learning Theory Conference, pp. 32–47. Springer, New York (2006)
2. Blum, A., Furst, M., Jackson, J., Kearns, M., Mansour, Y., Rudich, S.: Weakly learning DNF and characterizing statistical query learning using Fourier analysis. In: Proceedings of the 26th Annual ACM Symposium on Theory of Computing, pp. 253–262. Association for computing Machinery, New York (1994)
3. Bshouty, N.H., Jackson, J.C.: Learning DNF over the uniform distribution using a quantum example oracle. SIAM J. Comput. **28**, 1136–1153 (1999)
4. Bshouty, N.H., Jackson, J.C., Tamon, C.: More efficient PAC-learning of DNF with membership queries under the uniform distribution. J. Comput. Syst. Sci. **68**, 205–234 (2004)
5. Bshouty, N.H., Mossel, E., O'Donnell, R., Servedio, R.A.: Learning DNF from random walks. J. Comput. Syst. Sci. **71**, 250–265 (2005)
6. Feldman, V.: On attribute efficient and non-adaptive learning of parities and DNF expressions. In: 18th Annual Conference on Learning Theory, pp. 576–590. Springer-Verlag, Berlin Heidelberg (2005)

7. Jackson, J.: An efficient membership-query algorithm for learning DNF with respect to the uniform distribution. J. Comput. Syst. Sci. **55**, 414–440 (1997)
8. Jackson, J., Shamir, E., Shwartzman, C.: Learning with queries corrupted by classification noise. Discret. Appl. Math. **92**, 157–175 (1999)
9. Jackson, J.C.: An efficient membership-query algorithm for learning DNF with respect to the uniform distribution. In: 35th Annual Symposium on Foundations of Computer Science, pp. 42–53. IEEE Computer Society Press, Los Alamitos (1994)
10. Jackson, J.C.: The Harmonic Sieve: A Novel Application of Fourier Analysis to Machine Learning Theory and Practice. Ph. D. thesis, Carnegie Mellon University (1995)
11. Klivans, A.R., Servedio, R.A.: Boosting and hard-core set construction. Mach. Learn. **51**, 217–238 (2003)

Learning Heavy Fourier Coefficients of Boolean Functions

1989; Goldreich, Levin

LUCA TREVISAN
Department of Computer Science, University of California at Berkeley, Berkeley, CA, USA

Keywords and Synonyms

Error-control codes, Reed–Muller code

Problem Definition

The Hamming distance $d_H(y,z)$ between two binary strings y and z of the same length is the number of entries in which y and z disagree. A binary error-correcting code of minimum distance d is a mapping $C:\{0,1\}^k \to \{0,1\}^n$ such that for every two distinct inputs $x, x' \in \{0,1\}^k$, the encodings $C(x)$ and $C(x')$ have Hamming distance at least d. Error-correcting codes are employed to transmit information over noisy channels. If a sender transmits an encoding $C(x)$ of a message x via a noisy channel, and the recipient receives a corrupt bit string $y \neq C(x)$, then, provided that y differs from $C(x)$ in at most $(d-1)/2$ locations, the recipient can recover y from $C(x)$. The recipient can do so by searching for the string x that minimizes the Hamming distance between $C(x)$ and y: there can be no other string x' such that $C(x')$ has Hamming distance $(d-1)/2$ or smaller from y, otherwise $C(x)$ and $C(x')$ would be within Hamming distance $d-1$ or smaller, contradicting the above definition. The problem of recovering the message x from the corrupted encoding y is the *unique decoding* problem for the error-correcting code C. For the above-described scheme to be feasible, the decoding problem must be solvable via an efficient algorithm. These notions are due to Hamming [4].

Suppose that C is a code of minimum distance d, and such that there are pairs of encodings $C(x)$, $C(x')$ whose distance is exactly d. Furthermore, suppose that a communication channel is used that could make a number of errors larger than $(d-1)/2$. Then, if the sender transmits an encoded message using C, it is no longer possible for the recipient to uniquely reconstruct the message. If the sender, for example, transmits $C(x)$, and the recipient receives a string y that is at distance $d/2$ from $C(x)$ and at distance $d/2$ from $C(x')$, then, from the perspective of the recipient, it is equally likely that the original message was x or x'. If the recipient knows an upper bound e on the number of entries that the channel has corrupted, then, given the received string y, the recipient can at least compute the list of all strings x such that $C(x)$ and y differ in at most e locations. An error-correcting code $C:\{0,1\}^k \to \{0,1\}^n$ is (e,L)-list decodable if, for every string $y \in \{0,1\}^n$, the set $\{x \in \{0,1\}^k : d_H(C(x),y) \le e\}$ has cardinality at most L. The problem of reconstructing the list given y and e is the *list-decoding problem* for the code C. Again, one is interested in efficient algorithms for this problem. The notion of list-decoding is due to Elias [1].

A code $C:\{0,1\}^k \to \{0,1\}^n$ is a *Hadamard code* if every two encodings $C(x)$, $C(x')$ differ in precisely $n/2$ locations. In the Computer Science literature, it is common to use the term Hadamard code for a specific construction (the *Reed–Muller code of order 2*) that satisfies the above property. For a string $a \in \{0,1\}^k$, define the function $\ell_a : \{0,1\}^k \to \{0,1\}$ as

$$\ell_a(x) := \sum_i a_i x_i \bmod 2 .$$

Observe that, for $a \neq b$, the two functions ℓ_a and ℓ_b differ on precisely $(2^k)/2$ inputs. For $n = 2^k$, the code $H:\{0,1\}^k \to \{0,1\}^n$ maps a message $a \in \{0,1\}^k$ into the n-bit string which is the *truth-table of the function* ℓ_a. That is, if $b_1,\dots,b_n$ is an enumeration of the $n = 2^k$ elements of $\{0,1\}^k$, and $a \in \{0,1\}^k$ is a message, then the encoding $H(a)$ is the n-bit string that contains the value $\ell_a(b_i)$ in the i-th entry. Note that any two encodings $H(x)$, $H(x')$ differ in precisely $n/2$ entries, and so what was just defined is a Hadamard code. From now on, the term *Hadamard code* will refer exclusively to this construction.

It is known that the Hadamard code $H:\{0,1\}^k \to \{0,1\}^{2^k}$ is $(\frac{1}{2}-\epsilon, \frac{1}{4\epsilon^2})$-list decodable for every $\epsilon > 0$. The Goldreich–Levin results provide efficient list-decoding algorithm.

The following definition of the *Fourier spectrum* of a boolean function will be needed later to state an application of the Goldreich–Levin results to computational learning theory. For a string $a \in \{0,1\}^k$, define the function $\chi_a : \{0,1\}^k \to \{-1,+1\}$ as $\chi_a(x) := (-1)^{\ell_a(x)}$. Equivalently, $\chi_a(x) = (-1)^{\sum_i a_i x_i}$. For two functions $f, g : \{0,1\}^k \to \mathbb{R}$, define their *inner product* as

$$\langle f, g \rangle := \frac{1}{2^k} \sum_x f(x) \cdot g(x) \,.$$

Then it is easy to see that, for every $a \neq b$, $\langle \chi_a, \chi_b \rangle = 0$, and $\langle \chi_a, \chi_a \rangle = 1$. This means that the functions $\{\chi_a\}_{a \in \{0,1\}^k}$ form an orthonormal basis for the set of all functions $f : \{0,1\}^k \to \mathbb{R}$. In particular, every such function f can be written as a linear combination

$$f(x) = \sum_a \hat{f}(a) \chi_a(x)$$

where the coefficients $\hat{f}(a)$ satisfy $\hat{f}(a) = \langle f, \chi_a \rangle$. The coefficients $\hat{f}(a)$ are called the *Fourier coefficients* of the function f.

Key Results

Theorem 1 *There is a randomized algorithm GL that, given in input an integer k and a parameter $\epsilon > 0$, and given oracle access to a function $f : \{0,1\}^k \to \{0,1\}$, runs in time polynomial in $1/\epsilon$ and in k and outputs, with high probability over its internal coin tosses, a set $S \subseteq \{0,1\}^k$ that contains all the strings $a \in \{0,1\}^k$ such that ℓ_a and f agree on at least a $1/2 + \epsilon$ fraction of inputs.*

Theorem 1 is proved by Goldreich and Levin [3]. The result can be seen as a list-decoding for the Hadamard code $H : \{0,1\}^k \to \{0,1\}^{2^k}$; remarkably, the algorithm runs in time polynomial in k, which is poly-logarithmic in the length of the given corrupted encoding.

Theorem 2 *There is a randomized algorithm KM that given in input an integer k and parameters $\epsilon, \delta > 0$, and given oracle access to a function $f : \{0,1\}^k \to \{0,1\}$, runs in time polynomial in $1/\epsilon$, in $1/\delta$, and in k and outputs a set $S \subseteq \{0,1\}^k$ and a value $g(a)$ for each $a \in S$.*

With high probability over the internal coin tosses of the algorithm,

1 *S contains all the strings $a \in \{0,1\}^k$ such that $|\hat{f}(a)| \geq \epsilon$, and*
2 *For every $a \in S$, $|\hat{f}(a) - g(a)| \leq \delta$.*

Theorem 2 is proved by Kushilevitz and Mansour [5]; it is an easy consequence of the Goldreich–Levin algorithm.

Applications

There are two key applications of the Goldreich–Levin algorithm: one is to cryptography and the other is to computational learning theory.

Application in Cryptography

In cryptography, a *one-way permutation* is a family of functions $\{p_n\}_{n \geq 1}$ such that: (i) for every n, $p_n : \{0,1\}^n \to \{0,1\}^n$ is bijective, (ii) there is a polynomial time algorithm that, given $x \in \{0,1\}^n$, computes $p_n(x)$, and (iii) for every polynomial time algorithm A and polynomial q, and for every sufficiently large n,

$$\mathbb{P}_{x \sim \{0,1\}^n}[A(p_n(x)) = x] \leq \frac{1}{q(n)} \,.$$

That is, even though computing $p_n(x)$ given x is doable in polynomial time, the task of computing x given $p_n(x)$ is intractable. A *hard core predicate* for a one-way permutation $\{p_n\}$ is a family of functions $\{B_n\}_{n \geq 1}$ such that: (i) for every n, $B_n : \{0,1\}^n \to \{0,1\}$, (ii) there is a polynomial time algorithm that, given $x \in \{0,1\}^n$, computes $B_n(x)$, and (iii) for every polynomial time algorithm A and polynomial q, and for every sufficiently large n,

$$\mathbb{P}_{x \sim \{0,1\}^n}[A(p_n(x)) = B_n(x)] \leq \frac{1}{2} + \frac{1}{q(n)} \,.$$

That is, even though computing $B_n(x)$ given x is doable in polynomial time, the task of computing $B_n(x)$ given $p_n(x)$ is intractable.

Goldreich and Levin [3] use their algorithm to show that every one-way permutation has a hard-core predicate, as stated in the next theorem.

Theorem 3 *Let $\{p_n\}$ be a one-way permutation; define $\{p'_n\}$ such that $p'_{2n}(x, y) := p_n(x), y$ and let $B_{2n}(x, y) := \sum_i x_i y_i \bmod 2$. (For odd indices, let $p'_{2n+1}(z, b) := p'_{2n}(z)$ and $B_{2n+1}(z, b) := B_{2n}(z)$.)*

Then $\{p'_n\}$ is a one-way permutation and $\{B_n\}$ is a hard-core predicate for $\{p'_n\}$.

This result is used in efficient constructions of pseudorandom generators, pseudorandom functions, and private-key encryption schemes based on one-way permutations. The interested reader is referred to Chapter 3 in Goldreich's monograph [2] for more details.

There are also related applications in computational complexity theory, especially in the study of average-case complexity. See [7] for an overview.

Application in Computational Learning Theory

Loosely speaking, in computational learning theory one is given an unknown function $f : \{0,1\}^k \to \{0,1\}$ and one wants to compute, via an efficient randomized algorithm, a representation of a function $g : \{0,1\}^k \to \{0,1\}$ that agrees with f on most inputs. In the *PAC learning* model, one has access to f only via randomly sampled pairs $(x, f(x))$; in the model of *learning with queries*, instead, one can evaluate f at points of one's choice. Kushilevitz and Mansour [5] suggest the following algorithm: using the algorithm of Theorem 2, find a set S of large coefficients and approximations $g(a)$ of the coefficients $\hat{f}(a)$ for $a \in S$. Then define the function $g(x) = \sum_{a \in S} g(a)\chi_a(x)$. If the error caused by the absence of the smaller coefficients and the imprecision in the larger coefficient is not too large, g and f will agree on most inputs. (A technical point is that g as defined above is not necessarily a boolean function, but it can be easily "rounded" to be boolean.) Kushilevitz and Mansour show that such an approach works well for the class of functions f for which $\sum_a |\hat{f}(a)|$ is bounded, and they observe that functions of small *decision tree complexity* fall into this class. In particular, they derive the following result.

Theorem 4 *There is a randomized algorithm that, given in input parameters k, m, ε and δ, and given oracle access to a function $f : \{0,1\}^k \to \{0,1\}$ of decision tree complexity at most m, runs in time polynomial in k, m, $1/\epsilon$ and $\log 1/\delta$ and, with probability at least $1 - \delta$ over its internal coin tosses, outputs a circuit computing a function $g : \{0,1\}^k \to \{0,1\}$ that agrees with f on at least a $1 - \epsilon$ fraction of inputs.*

Another application of the Kushilevitz–Mansour technique is due to Linial, Mansour, and Nisan [6].

Cross-References

► Decoding Reed–Solomon Codes

Recommended Reading

1. Elias, P.: List decoding for noisy channels. Technical Report 335, Research Laboratory of Electronics, MIT, Campridge, MA, USA (1957)
2. Goldreich, O.: The Foundations of Cryptography – Volume 1. Cambridge University Press, Campridge, UK (2001)
3. Goldreich, O., Levin, L.: A hard-core predicate for all one-way functions. In: Proceedings of the 21st ACM Symposium on Theory of Computing, pp. 25–32 Seattle, 14–17 May 1989
4. Hamming, R.: Error detecting and error correcting codes. Bell Syst. Tech. J. **29**, 147–160 (1950)
5. Kushilevitz, E., Mansour, Y.: Learning decision trees using the fourier spectrum. SIAM J. Comp. **22**(6), 1331–1348 (1993)
6. Linial, N., Mansour, Y., Nisan, N.: Constant depth circuits, fourier transform and learnability. J. ACM **40**(3), 607–620 (1993)
7. Trevisan, L.: Some applications of coding theory in computational complexity. Quaderni Matematica **13**, 347–424 (2004) arXiv:cs.CC/0409044

Learning with Malicious Noise

1993; Kearns, Li

PETER AUER
Institute for Computer Science, University of Leoben, Leoben, Austria

Problem Definition

This problem is concerned with PAC learning of concept classes when training examples are effected by malicious errors. The PAC (probably approximately correct) model of learning (also known as the distribution-free model of learning) was introduced by Valiant [11]. This model makes the idealized assumption that error-free training examples are generated from the same distribution which is then used to evaluate the learned hypothesis. In many environments, however, there is some chance that an erroneous example is given to the learning algorithm. The malicious noise model – again introduced by Valiant [12] – extends the PAC model by allowing example errors of any kind: it makes no assumptions on the nature of the errors that occur. In this sense the malicious noise model is a worst-case model of errors, in which errors may be generated by an adversary whose goal is to foil the learning algorithm. Kearns and Li [7,8] study the maximal malicious error rate such that learning is still possible. They also provide a canonical method to transform any standard learning algorithm into an algorithm which is robust against malicious noise.

Notations

Let X be a set of instances. The goal of a learning algorithm is to infer an unknown subset $C \subseteq X$ of instances which exhibit a certain property. Such subsets are called concepts. It is known to the learning algorithm that the correct concept C is from a concept class $\mathcal{C} \subseteq 2^X$, $C \in \mathcal{C}$. Let $C(x) = 1$ if $x \in C$ and $C(x) = 0$ if $x \notin C$. As input the learning algorithm receives an accuracy parameter $\varepsilon > 0$, a confidence parameter $\delta > 0$, and the malicious noise rate $\beta \geq 0$. The learning algorithm may request a sample of labeled instances $S = \langle (x_1, \ell_1), \dots, (x_m, \ell_m) \rangle$, $x_i \in X$ and $\ell_i \in \{0,1\}$, and produces a hypothesis $H \subseteq X$. Let $\mathcal{D}$ be the unknown distribution of instances in X. Learning is successful if H misclassifies an example with probability less than ε, $\mathrm{err}_{\mathcal{D}}(C,H) := \mathcal{D}\{x \in X : C(x) \neq H(x)\} < \varepsilon$. A learning algorithm is requested to be successful with probability $1-\delta$. The error of a hypothesis H in respect to a sample S of labeled instances is defined as $\mathrm{err}(S,H) := |\{(x,\ell) \in S : H(x) \neq \ell\}|/|S|$.

The VC-dimension $\mathrm{VC}(C)$ of a concept class C is the maximal number of instances $x_1,\ldots,x_d$ such that $\{(C(x_1),\ldots,C(x_d)) : C \in C\} = \{0,1\}^d$. The VC-dimension is a measure for the difficulty to learn concept class C [3].

To investigate the computational complexity of learning algorithms, sequences of concept classes with increasing complexity $(X_n, C_n)_n = \langle (X_1,C_1),(X_2,C_2),\ldots\rangle$ are considered. In this case the learning algorithm receives also a complexity parameter n as input.

Generation of Examples

In the malicious noise model the labeled instances (x_i,ℓ_i) are generated independently from each other by the following random process.

(a) Correct examples: with probability $1-\beta$ an instance x_i is drawn from distribution $\mathcal{D}$ and labeled by the correct concept C, $\ell_i = C(x_i)$.
(b) Noisy examples: with probability β an arbitrary example (x_i,ℓ_i) is generated, possibly by an adversary.

Problem 1 (Malicious Noise Learning of (X,C))
INPUT: *Reals* $\varepsilon,\delta > 0$, $\beta \geq 0$.
OUTPUT: *A hypothesis* $H \subseteq X$.
For any distribution $\mathcal{D}$ *on* X *and any concept* $C \in C$, *the algorithm needs to produce with probability* $1-\delta$ *a hypothesis* H *such that* $\mathrm{err}_{\mathcal{D}}(C,H) < \varepsilon$. *The probability* $1-\delta$ *is taken in respect to the random sample* $(x_1,\ell_1),\ldots,(x_m,\ell_m)$ *requested by the algorithm. The examples* (x_i,ℓ_i) *are generated as defined above.*

Problem 2 (Polynomial Malicious Noise Learning of $(X_n,C_n)_n$)
INPUT: *Reals* $\varepsilon,\delta > 0$, $\beta \geq 0$, *integer* $n \geq 1$.
OUTPUT: *A hypothesis* $H \subseteq X_n$.
For any distribution $\mathcal{D}$ *on* X_n *and any concept* $C \in C_n$, *the algorithm needs to produce with probability* $1-\delta$ *a hypothesis* H *such that* $\mathrm{err}_{\mathcal{D}}(C,H) < \varepsilon$. *The computational complexity of the algorithm must be bounded by a polynomial in* $1/\varepsilon$, $1/\delta$, *and* n.

Key Results

Theorem 1 ([8]) *If there are concepts* $C_1,C_2 \in C$ *and instances* $x_1,x_2 \in X$ *such that* $C_1(x_1) = C_1(x_2)$ *and* $C_2(x_1) \neq C_2(x_2)$, *then no algorithm learns* C *with malicious noise rate* $\beta \geq \varepsilon/(1+\varepsilon)$.

Theorem 2 *Let* $\Delta > 0$ *and* $d = \mathrm{VC}(C)$. *For a suitable constant* κ, *any algorithm which requests a sample* S *of* $m \geq \kappa(\varepsilon d \log 1/(\Delta\delta))/\Delta^2$ *labeled examples and returns a hypothesis* $H \in C$ *which minimizes* $\mathrm{err}(S,H)$, *learns the concept class* C *with malicious noise rate* $\beta \leq \varepsilon/(1+\varepsilon) - \Delta$.

Lower bounds on the number of examples necessary for learning with malicious noise were derived by Cesa-Bianchi et al. [5].

Theorem 3 ([5]) *Let* $\Delta > 0$ *and* $d = \mathrm{VC}(C) \geq 3$. *There is a constant* κ, *such that any algorithm which learns* C *with malicious noise rate* $\beta = \varepsilon/(1+\varepsilon) - \Delta$ *by requesting a sample and returning a hypothesis* $H \in C$ *which minimizes* $\mathrm{err}(S,H)$, *needs a sample of size at least* $m \geq \kappa\varepsilon d/\Delta^2$.

A general conversion of a learning algorithm for the noise-free model into an algorithm for the malicious noise model was given by Kearns and Li.

Theorem 4 ([8]) *Let* $\mathcal{A}$ *be a (polynomial-time) learning algorithm which learns concept classes* C_n *from* $m(\varepsilon,\delta,n)$ *noise-free examples, i. e.* $\beta = 0$. *Then* $\mathcal{A}$ *can be converted into a (polynomial-time) learning algorithm for* C_n *for any malicious noise rate* $\beta \leq \log m(\varepsilon/8, 1/2, n)/m(\varepsilon/8, 1/2, n)$.

The next theorem relates learning with malicious noise to a type of combinatorial optimization problems.

Theorem 5 ([8]) *Let* $r \geq 1$ *and* $\alpha > 1$.

1. *Let* $\mathcal{A}$ *be an algorithm which, for any sample* S, *returns a hypothesis* $H \in C$ *with* $\mathrm{err}(S,H) \leq r \cdot \min_{C \in C} \mathrm{err}(S,C)$. *Then* $\mathcal{A}$ *learns concept class* C *for any malicious noise rate* $\beta \leq \varepsilon/(\alpha(1+\varepsilon)r)$ *from a sufficiently large sample.*
2. *Let* $\mathcal{A}$ *be a polynomial-time learning algorithm for concept classes* C_n *which tolerates a malicious noise rate* $\beta = \varepsilon/r$. *Then* $\mathcal{A}$ *can be converted into a polynomial-time algorithm which for any sample* S, *with high probability returns a hypothesis* $H \in C_n$ *such that* $\mathrm{err}(S,H) \leq \alpha r \cdot \min_{C \in C} \mathrm{err}(S,C)$.

The computational hardness of several such related combinatorial optimization problems was shown by Ben-David, Eiron, and Long [2].

Applications

Several extensions of the learning model with malicious noise have been proposed, in particular the agnostic learning model [9] and the statistical query model [6]. Following relations between these models and the malicious noise model have been established.

Theorem 6 ([9]) *If concept class C is polynomial-time learnable in the agnostic model, then C is polynomial-time learnable with any malicious noise rate $\beta \leq \varepsilon/2$.*

Theorem 7 ([6]) *If C is learnable from (relative error) statistical queries, then C is learnable with any malicious noise rate $\beta \leq \varepsilon/\log^p(1/\varepsilon)$ for a suitable large p independent of C.*

Another learning model related to the malicious noise model is learning with nasty noise [4]. In this model examples effected by malicious noise are not chosen at random with probability β, but an adversary might manipulate an arbitrary fraction of βm examples out of a given sample of size m. The malicious noise model was also considered in the context of on-line learning [1] and boosting [10].

Cross References

- ► Boosting Textual Compression
- ► PAC Learning
- ► Perceptron Algorithm
- ► Statistical Query Learning

Recommended Reading

1. Auer, P., Cesa-Bianchi, N.: On-line learning with malicious noise and the closure algorithm. Ann. Math. Artif. Intell. **23**, 83–99 (1998)
2. Ben-David, S., Eiron, N., Long, P.: On the difficulty of approximately maximizing agreements. J. CSS **66**, 496–514 (2003)
3. Blumer, A., Ehrenfeucht, A., Haussler, D., Warmuth, M.: Learnability and the Vapnik-Chervonenkis dimension. J. ACM **36**, 929–965 (1989)
4. Bshouty, N., Eiron, N., Kushilevitz, E.: PAC learning with nasty noise. TCS **288**, 255–275 (2002)
5. Cesa-Bianchi, N., Dichterman, E., Fischer, P., Shamir, E., Simon, H.U.: Sample-efficient strategies for learning in the presence of noise. J. ACM **46**, 684–719 (1999)
6. Aslam, J.A., Decatur, S.E.: Specification and simulation of statistical query algorithms for efficiency and noise tolerance. J. CSS **56**, 191–208 (1998)
7. Kearns, M., Li, M.: Learning in the presence of malicious errors. In: Proc. 20th ACM Symp. Theory of Computing, pp. 267–280, Chicago, 2–4 May 1988
8. Kearns, M., Li, M.: Learning in the presence of malicious errors. SIAM J. Comput. **22**, 807–837 (1993)
9. Kearns, M., Schapire, R., Sellie, L.: Toward efficient agnostic learning. Mach. Learn. **17**, 115–141 (1994)
10. Servedio, R.A.: Smooth boosting and learning with malicious noise. JMLR **4**, 633–648 (2003)
11. Valiant, L.: A theory of the learnable. C. ACM **27**, 1134–1142 (1984)
12. Valiant, L.: Learning disjunctions of conjunctions. In: Proc. 9th Int. Joint Conference on Artificial Intelligence, pp. 560–566, Los Angeles, August 1985

Learning Significant Fourier Coefficients over Finite Abelian Groups

2003; Akavia, Goldwasser, Safra

ADI AKAVIA
Department of Electrical Engineering and Computer Science, MIT, Cambridge, MA, USA

Keywords and Synonyms

Learning heavy fourier coefficients; Finding heavy fourier coefficients

Problem Definition

Fourier transform is among the most widely used tools in computer science. Computing the Fourier transform of a signal of length N may be done in time $\Theta(N \log N)$ using the Fast Fourier Transform (FFT) algorithm. This time bound clearly cannot be improved below $\Theta(N)$, because the output itself is of length N. Nonetheless, it turns out that in many applications it suffices to find only the *significant Fourier coefficients*, i. e., Fourier coefficients occupying, say, at least 1% of the energy of the signal. This motivates the problem discussed in this entry: the problem of efficiently finding and approximating the significant Fourier coefficients of a given signal (SFT, in short). A naive solution for SFT is to first compute the entire Fourier transform of the given signal and then to output only the significant Fourier coefficients; thus yielding no complexity improvement over algorithms computing the entire Fourier transform. In contrast, SFT can be solved far more efficiently in running time $\widetilde{\Theta}(\log N)$ and while reading at most $\widetilde{\Theta}(\log N)$ out of the N signal's entries [2]. This fast algorithm for SFT opens the way to applications taken from diverse areas including computational learning, error correcting codes, cryptography, and algorithms.

It is now possible to formally define the SFT problem, restricting our attention to discrete signals. Use functional notation where a signal is a function $f\colon G \to \mathbb{C}$ over a finite Abelian group G, its energy is $\|f\|_2^2 \overset{\text{def}}{=} 1/|G| \sum_{x \in G} |f(x)|^2$, and its maximal amplitude is $\|f\|_\infty \overset{\text{def}}{=} \max\{|f(x)| \,|x \in G\}$.[1] For ease of presentation

[1]For readers more accustomed to vector notation, the authors remark that there is a simple correspondence between vector and functional notation. For example, a one-dimensional signal $(v_1, \ldots, v_N) \in \mathbb{C}^{\mathbb{N}}$ corresponds to the function $f\colon \mathbb{Z}_N \to \mathbb{C}$ defined by $f(i) = v_i$ for all $i = 1, \ldots, N$. Likewise, a two-dimensional signal $M \in \mathbb{C}^{N_1 \times N_2}$ corresponds to the function $f\colon \mathbb{Z}_{N_1} \times \mathbb{Z}_{N_2} \to \mathbb{C}$ defined by $f(i,j) = M_{ij}$ for all $i = 1, \ldots, N_1$ and $j = 1, \ldots, N_2$.

assume without loss of generality that $G = \mathbb{Z}_{N_1} \times \mathbb{Z}_{N_2} \times \cdots \times \mathbb{Z}_{N_k}$ for $N_1, \ldots, N_k \in \mathbb{Z}^+$ (i. e., positive integers), and for $\mathbb{Z}_N$ is the additive group of integers modulo N.

The *Fourier transform* of f is the function $\widehat{f}: G \to \mathbb{C}$ defined for each $\alpha = (\alpha_1, \ldots, \alpha_k) \in G$ by

$$\widehat{f}(\alpha) \overset{\text{def}}{=} \frac{1}{|G|} \sum_{(x_1,\ldots,x_k)\in G} \left[f(x_1, \ldots, x_k) \prod_{j=1}^{k} \omega_{N_j}^{\alpha_j x_j} \right],$$

where $\omega_{N_j} = \exp(i2\pi/N_j)$ is a primitive root of unity of order N_j. For any $\alpha \in G$, $val_\alpha \in \mathbb{C}$ and $\tau, \varepsilon \in [0, 1]$, say that α is a *τ-significant Fourier coefficient* iff $|\widehat{f}(\alpha)|^2 \geq \tau\|f\|_2^2$, and say that val_α is an *ε-approximation for $\widehat{f}(\alpha)$* iff $|val_\alpha - \widehat{f}(\alpha)| < \varepsilon$.

Problem 1 (SFT)
INPUT: *Integers $N_1, \ldots, N_k \geq 2$ specifying the group $G = \mathbb{Z}_{N_1} \times \cdots \times \mathbb{Z}_{N_k}$, a threshold $\tau \in (0, 1)$, an approximation parameter $\varepsilon \in (0, 1)$, and oracle access[2] to $f: G \to \mathbb{C}$.*
OUTPUT: *A list of all τ-significant Fourier coefficients of f along with ε-approximations for them.*

Key Results

The key result of this entry is an algorithm solving the SFT problem which is much faster than algorithms for computing the entire Fourier transform. For example, for f a Boolean function over $\mathbb{Z}_N$, the running time of this algorithm is $\log N \cdot poly(\log\log N, 1/\tau, 1/\varepsilon)$, in contrast to the $\Theta(N \log N)$ running time of the FFT algorithm. This algorithm is named the SFT algorithm.

Theorem 1 (SFT algorithm [2]) *There is an algorithm solving the SFT problem with running time $\log|G| \cdot poly(\log\log|G|, \|f\|_\infty/\|f\|_2, 1/\tau, 1/\varepsilon)$ for $|G| = \prod_{j=1}^{k} N_j$ the cardinality of G.*

Remarks
1. The above result extends to functions f over any finite Abelian group G, as long as the algorithm is given a description of G by its generators and their orders [2].
2. The SFT algorithm reads at most $\log|G| \cdot poly(\log\log|G|, \|f\|_\infty/\|f\|_2, 1/\tau, 1/\varepsilon)$ out of the $|G|$ values of the signal.
3. The SFT algorithm is non adaptive, that is, oracle queries to f are independent of the algorithm's progress.
4. The SFT algorithm is a probabilistic algorithm having a small error probability, where probability is taken over the internal coin tosses of the algorithm. The error probability can be made arbitrarily small by standard amplification techniques.

The SFT algorithm follows a line of works solving the SFT problem for restricted function classes. Goldreich and Levin [9] gave an algorithm for Boolean functions over the group $\mathbb{Z}_2^k = \{0, 1\}^k$. The running time of their algorithm is polynomial in $k, 1/\tau$ and $1/\varepsilon$. Mansour [10] gave an algorithm for complex functions over groups $G = \mathbb{Z}_{N_1} \times \cdots \times \mathbb{Z}_{N_k}$ provided that $N_1, \ldots, N_k$ are powers of two. The running time of his algorithm is polynomial in $\log|G|, \log(\max_{\alpha\in G}|\widehat{f}(\alpha)|), 1/\tau$ and $1/\varepsilon$. Gilbert et al. [6] gave an algorithm for complex functions over the group $\mathbb{Z}_N$ for any positive integer N. The running time of their algorithm is polynomial in $\log N$, $\log(\max_{x\in\mathbb{Z}_N} f(x)/\min_{x\in\mathbb{Z}_N} f(x)), 1/\tau$ and $1/\varepsilon$. Akavia et al. [2] gave an algorithm for complex functions over any finite Abelian group. The latter [2] improves on [6] even when restricted to functions over $\mathbb{Z}_N$ in achieving $\log N \cdot poly(\log\log N)$ rather than $poly(\log N)$ dependency on N. Subsequent works [7] improved the dependency of [6] on τ and ε.

Applications

Next, the paper surveys applications of the SFT algorithm [2] in the areas of computational learning theory, coding theory, cryptography, and algorithms.

Applications in Computational Learning Theory

A common task in computational learning is to find a hypothesis h approximating a function f, when given only samples of the function f. Samples may be given in a variety of forms, e. g., via oracle access to f. We consider the following variant of this learning problem: f and h are complex functions over a finite Abelian group $G = \mathbb{Z}_{N_1} \times \cdots \times \mathbb{Z}_{N_k}$, the goal is to find h such that $\|f - h\|_2^2 \leq \gamma\|f\|_2^2$ for $\gamma > 0$ an approximation parameter, and samples of f are given via oracle access.

A straightforward application of the SFT algorithm gives an efficient solution to the above learning problem, provided that there is a small set $\Gamma \subseteq G$ s.t. $\sum_{\alpha\in\Gamma} |\widehat{f}(\alpha)|^2 > (1 - \gamma/3)\|f\|_2^2$. The learning algorithm operates as follows. It first runs the SFT algorithm to find all $\alpha = (\alpha_1, \ldots, \alpha_k) \in G$ that are $\gamma/|\Gamma|$-significant Fourier coefficients of f along with their $\gamma/|\Gamma|\|f\|_\infty$-

[2] Say that an algorithm is given *oracle access* to a function f over G, if it can request and receive the value $f(x)$ for any $x \in G$ in unit time.

approximations val_α, and then returns the hypothesis

$$h(x_1,\ldots,x_k) \overset{\text{def}}{=} \sum_{\alpha \text{ is } \gamma/|\Gamma|-\text{significant}} val_\alpha \cdot \prod_{j=1}^{k} \omega_{N_j}^{\alpha_j x_j} .$$

This hypothesis h satisfies that $\|f-h\|_2^2 \le \gamma\|f\|_2^2$. The running time of this learning algorithm and the number of oracle queries it makes is polynomially bounded by $\log|G|$, $\|f\|_\infty/\|f\|_2$, $|\Gamma|\|f\|_\infty/\gamma$.

Theorem 2 *Let $f\colon G \to \mathbb{C}$ be a function over $G = \mathbb{Z}_{N_1} \times \cdots \times \mathbb{Z}_{N_k}$, and $\gamma > 0$ an approximation parameter. Denote $t = \min\{|\Gamma| \mid \Gamma \subseteq G \text{ s.t. } \sum_{\alpha\in\Gamma} |\widehat{f}(\alpha)|^2 > (1-\gamma/3)\|f\|_2^2\}$. There is an algorithm that given $N_1,\ldots,N_k$, γ, and oracle access to f, outputs a (short) description of $h\colon G \to \mathbb{C}$ s.t. $\|f-h\|_2^2 < \gamma\|f\|_2^2$. The running time of this algorithm is $\log|G| \cdot poly(\log\log|G|, \|f\|_\infty/\|f\|_2, t\|f\|_\infty/\gamma)$.*

More examples of function classes that can be efficiently learned using our SFT algorithm are given in [3].

Applications in Coding Theory

Error correcting codes encode messages in a way that allows *decoding*, that is, recovery of the original message, even in the presence of noise. When the noise is very high, unique decoding may be infeasible, nevertheless it may still be possible to *list decode*, that is, to find a short list of messages containing the original message. Codes equipped with an efficient list decoding algorithm have found many applications (see [11] for a survey).

Formally, a binary code is a subset $C \subseteq \{0,1\}^*$ of *codewords* each encoding some message. Denote by $C_{N,x} \in \{0,1\}^N$ a codeword of length N encoding a message x. The *normalized Hamming distance* between a codeword $C_{N,x}$ and a received word $w \in \{0,1\}^N$ is $\Delta(C_{N,x}, w) \overset{\text{def}}{=} 1/N|\{i \in \mathbb{Z}_N \mid C_{N,x}(i) \neq w(i)\}|$ where $C_{N,x}(i)$ and $w(i)$ are the ith bits of $C_{N,x}$ and w, respectively. Given $w \in \{0,1\}^N$ and a noise parameter $\eta > 0$, the *list decoding* task is to find a list of all messages x such that $\Delta(C_{N,x}, w) < \eta$. The received word w may be given explicitly or implicitly; we focus on the latter where oracle access to w is given. Goldreich and Levin [9] give a list decoding algorithm for Hadamard codes, using in a crucial way their algorithm solving the SFT problem for functions over the Boolean cube.

The SFT algorithm for functions over ZZ_N is a key component in a list decoding algorithm given by Akavia et al. [2]. This list decoding algorithm is applicable to a large class of codes. For example, it is applicable to the code $C^{msb} = \{C_{N,x}\colon \mathbb{Z}_N \to \{0,1\}\}_{x\in\mathbb{Z}_N^*, N\in\mathbb{Z}^+}$ whose codewords are $C_{N,x}(j) = msb_N(j \cdot x \mod N)$ for $msb_N(y) = 1$ iff $y \ge N/2$ and $msb_N(y) = 0$ otherwise. More generally, this list decoding algorithm is applicable to any *Multiplication code* $C^{\mathcal{P}}$ for $\mathcal{P}$ a family of *balanced* and *well concentrated* functions, as defined below. The running time of this list decoding algorithm is polynomial in $\log N$ and $1/(1-2\eta)$, as long as $\eta < \frac{1}{2}$.

Abstractly, the list decoding algorithm of [2] is applicable to any code that is "balanced," "(well) concentrated," and "recoverable," as defined next (and those Fourier coefficients have small greatest common divisor (GCD) with N). A code is *balanced* if $\Pr_{j\in\mathbb{Z}_N}[C_{N,x}(j) = 0] = \Pr_{j\in\mathbb{Z}_N}[C_{N,x}(j) = 1]$ for every codeword $C_{N,x}$. A code is *(well) concentrated* if its codewords can be approximated by a small number of significant coefficients in their Fourier representation (and those Fourier coefficients have small greatest common divisor (GCD) with N). A code is *recoverable* if there is an efficient algorithm mapping each Fourier coefficient α to a short list of codewords for which α is a significant Fourier coefficient. The key property of concentrated codes is that received words w share a significant Fourier coefficient with all close codewords $C_{N,x}$. The high level structure of the list decoding algorithm of [2] is therefore as follows. First it runs the SFT algorithm to find all significant Fourier coefficients α of the received word w. Second for each such α, it runs the recovery algorithm to find all codewords $C_{N,x}$ for which α is significant. Finally, it outputs all those codewords $C_{N,x}$.

Definition 1 (Multiplication codes [2]) Let $\mathcal{P} = \{P_N\colon \mathbb{Z}_N \to \{0,1\}\}_{N\in\mathbb{Z}^+}$ be a family of functions. Say that $C^{\mathcal{P}} = \{C_{N,x}\colon \mathbb{Z}_N \to \{0,1\}\}_{x\in\mathbb{Z}_N^*, N\in\mathbb{Z}^+}$ is a *multiplication code for $\mathcal{P}$* if for every $N \in \mathbb{Z}^+$ and $x \in \mathbb{Z}_N^*$, the encoding $C_{N,x}\colon \mathbb{Z}_N \to \{0,1\}$ of x is defined by

$$C_{N,x}(j) = P(j \cdot x \bmod N) .$$

Definition 2 (Well concentrated [2]) Let $\mathcal{P} = \{P_N\colon \mathbb{Z}_N \to \mathbb{C}\}_{N\in\mathbb{Z}^+}$ be a family of functions. Say that $\mathcal{P}$ is *well concentrated* if $\forall N \in \mathbb{Z}^+, \gamma > 0, \exists \Gamma \subseteq \mathbb{Z}_N$ s.t. (i) $|\Gamma| \le poly(\log N/\gamma)$, (ii) $\sum_{\alpha\in\Gamma} |\widehat{P_N}(\alpha)|^2 \ge (1-\gamma)\|P_N\|_2^2$, and (iii) for all $\alpha \in \Gamma$, $gcd(\alpha, N) \le poly(\log N/\gamma)$ (where $gcd(\alpha, N)$ is the greatest common divisor of α and N).

Theorem 3 (List decoding [2]) *Let $\mathcal{P} = \{P_N\colon \mathbb{Z}_N \to \{0,1\}\}_{N\in\mathbb{Z}^+}$ be a family of efficiently computable[3], well concentrated, and balanced functions. Let $C^{\mathcal{P}}$ =*

[3] $\mathcal{P} = \{P_N : \mathbb{Z}_N \to \{0,1\}\}_{N\in\mathbb{Z}^+}$ is a family of *efficiently computable* functions if there is an algorithm that given any $N \in \mathbb{Z}^+$ and $x \in \mathbb{Z}_N$ outputs $P_N(x)$ in time $poly(\log N)$.

$\{C_{N,x} : \mathbb{Z}_N \to \{0,1\}\}_{x \in \mathbb{Z}_N^*, N \in \mathbb{Z}^+}$ *be the multiplication code for* $\mathcal{P}$. *Then there is an algorithm that, given* $N \in \mathbb{Z}_N^+$, $\eta < \frac{1}{2}$ *and oracle access to* $w : \mathbb{Z}_N \to \{0,1\}$, *outputs all* $x \in \mathbb{Z}_N^*$ *for which* $\Delta(C_{N,x}, w) < \eta$. *The running time of this algorithm is polynomial in* $\log N$ *and* $1/(1-2\eta)$.

Remarks

1. The requirement that $\mathcal{P}$ is a family of *efficiently computable* functions can be relaxed. It suffices to require that the list decoding algorithm receives or computes a set $\Gamma \subseteq \mathbb{Z}_N$ with properties as specified in Definition 2.
2. The requirement that $\mathcal{P}$ is a family of *balanced* functions can be relaxed. Denote $bias(\mathcal{P}) = \min_{b \in \{0,1\}} \inf_{N \in \mathbb{Z}^+} \Pr_{j \in \mathbb{Z}_N}[P_N(j) = b]$. Then the list decoding algorithm of [2] is applicable to $C^{\mathcal{P}}$ even when $bias(\mathcal{P}) \neq \frac{1}{2}$, as long as $\eta < bias(\mathcal{P})$.

Applications in Cryptography

Hard-core predicates for one-way functions are a fundamental cryptographic primitive, which is central for many cryptographic applications such as pseudo-random number generators, semantic secure encryption, and cryptographic protocols. Informally speaking, a Boolean predicate P is a *hard-core predicate* for a function f if $P(x)$ is easy to compute when given x, but hard to guess with a non-negligible advantage beyond 50% when given only $f(x)$. The notion of hardcore predicates was introduced by Blum and Micali [2]. Goldreich and Levin [9] showed a randomized hardcore predicate for any one-way function, using in a crucial way their algorithm solving the SFT problem for functions over the Boolean cube.

Akavia et al. [2] introduce a unifying framework for proving that a predicate P is hard-core for a one-way function f. Applying their framework they prove for a wide class of predicates—*segment predicates*—that they are hard-core predicates for various well-known candidate one-way functions. Thus showing new hard-core predicates for well-known one-way function candidates as well as reproving old results in an entirely different way.

Elaborating on the above, a *segment predicate* is any assignment of Boolean values to an arbitrary partition of $\mathbb{Z}_N$ into $poly(\log N)$ segments, or dilations of such an assignment. Akavia et al. [2] prove that any segment predicate is hard-core for any one-way function f defined over $\mathbb{Z}_N$ for which, for a non-negligible fraction of the x's in $\mathbb{Z}_N$, given $f(x)$ and y, one can efficiently compute $f(xy)$ (where xy is multiplication in $\mathbb{Z}_N$). This includes the following functions: the *exponentiation* function $EXP_{p,g} : \mathbb{Z}_p \to \mathbb{Z}_p^*$ defined by $EXP_{p,g}(x) = g^x \bmod p$ for each prime p and a generator g of the group $\mathbb{Z}_p^*$; the *RSA* function $RSA : \mathbb{Z}_N^* \to \mathbb{Z}_N^*$ defined by $RSA(x) = e^x \bmod N$ for each $N = pq$ a product of two primes p, q, and e co-prime to N; the *Rabin* function $Rabin : \mathbb{Z}_N^* \to \mathbb{Z}_N^*$ defined by $Rabin(x) = x^2 \bmod N$ for each $N = pq$ a product of two primes p, q; and the *elliptic curve log* function defined by $ECL_{a,b,p,Q} = xQ$ for each elliptic curve $E_{a,b,p}(Z_p)$ and Q a point of high order on the curve.

The SFT algorithm is a central tool in the framework of [2]: Akavia et al. take a list decoding methodology, where computing a hard-core predicate corresponds to computing an entry in some error correcting code, predicting a predicate corresponds to access to an entry in a corrupted codeword, and the task of inverting a one-way function corresponds to the task of list decoding a corrupted codeword. The codes emerging in [2] are multiplication codes (see Definition 1 above), which are list decoded using the SFT algorithm.

Definition 3 (Segment predicates [2]) Let $\mathcal{P} = \{P_N : \mathbb{Z}_N \to \{0,1\}\}_{N \in \mathbb{Z}^+}$ be a family of predicates that are non-negligibly far from constant[4].

- It can be sayed that P_N is a basic t-segment predicate *if* $P_N(x+1) \neq P_N(x)$ *for at most t x's in* $\mathbb{Z}_N$.
- It can be sayed that P_N is a t-segment predicate *if there exist a basic t-segment predicate* P' *and* $a \in \mathbb{Z}_N$ *which is co-prime to N s.t.* $\forall x \in \mathbb{Z}_N, P_N(x) = P'(x/a)$.
- It can be sayed that $\mathcal{P}$ is a family of segment predicates *if* $\forall N \in \mathbb{Z}^+$, P_N *is a t(N)-segment predicate for* $t(N) \leq poly(\log N)$.

Theorem 4 (Hardcore predicates [2]) *Let* $\mathcal{P}$ *be a family of segment predicates. Then,* $\mathcal{P}$ *is hard-core for RSA, Rabin, EXP, ECL, under the assumption that these are one-way functions.*

Application in Algorithms

Our modern times are characterized by information explosion incurring a need for faster and faster algorithms. Even algorithms classically regarded as efficient—such as the FFT algorithm with its $\Theta(N \log N)$ complexity—are often too slow for data-intensive applications, and linear or even sub-linear algorithms are imperative. Despite the vast variety of fields and applications where algorithmic challenges arise, some basic algorithmic building blocks emerge in many of the existing algorithmic solutions. Accelerating such building blocks can therefore accelerate

[4]A family of functions $\mathcal{P} = \{P_N : \mathbb{Z}_N \to \{0,1\}\}_{N \in \mathbb{Z}^+}$ is *non-negligibly far from constant* if $\forall N \in \mathbb{Z}^+$ and $b \in \{0,1\}$, $\Pr_{j \in \mathbb{Z}_N}[P_N(j) = b] \leq 1 - poly(1/\log N)$.

many existing algorithms. One of these recurring building blocks is the Fast Fourier Transform (FFT) algorithm. The SFT algorithm offers a great efficiency improvement over the FFT algorithm for applications where it suffices to deal only with the significant Fourier coefficients. In such applications, replacing the FFT building block with the SFT algorithm accelerates the $\Theta(N \log N)$ complexity in each application of the FFT algorithm to $poly(\log N)$ complexity [1]. Lossy compression is an example of such an application [1,5,8]. To elaborate, central component in several transform compression methods (e. g., JPEG) is to first apply Fourier (or Cosine) transform to the signal, and then discard many of its coefficients. To accelerate such algorithms —instead of computing the entire Fourier (or Cosine) transform—the SFT algorithm can be used to directly approximate only the significant Fourier coefficients. Such an accelerated algorithm achieves compression guarantee as good as the original algorithm (and possibly better), but with running time improved to $poly(\log N)$ in place of the former $\Theta(N \log N)$.

Cross References

- ▶ Abelian Hidden Subgroup Problem
- ▶ Learning Constant-Depth Circuits
- ▶ Learning DNF Formulas
- ▶ Learning Heavy Fourier Coefficients of Boolean Functions
- ▶ Learning with Malicious Noise
- ▶ List Decoding near Capacity: Folded RS Codes
- ▶ PAC Learning
- ▶ Statistical Query Learning

Recommended Reading

1. Akavia, A., Goldwasser, S.: Manuscript submitted as an NSF grant, awarded (2005) CCF-0514167
2. Akavia, A., Goldwasser, S., Safra, S.: Proving hard-core predicates using list decoding. In: Proceedings of the 44th Symposium on Foundations of Computer Science (FOCS'03), pp. 146–157. IEEE Computer Society (2003)
3. Atici, A., Servedio, R.A.: Learning unions of $\omega(1)$-dimensional rectangles. In: ALT, pp. 32–47 (2006)
4. Blum, M., Micali, S.: How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits. SIAM J. Comput. **4**(13), 850–864 (1984)
5. Cormode, G., Muthukrishnan, S.: Combinatorial algorithms for compressed sensing. In: Structural Information and Communication Complexity, 13th International Colloquium, SIROCCO (2006), Chester, UK, July 2–5, 2006 pp. 280–294
6. Gilbert, A.C., Guha, S., Indyk, P., Muthukrishnan, S., Strauss, M.: Near-optimal sparse fourier representations via sampling. In: Proceedings of the thiry-fourth annual ACM symposium on Theory of computing, pp. 152–161. ACM Press (2002)
7. Gilbert, A.C., Muthukrishnan, S., Strauss, M.J.: Improved time bounds for near-optimal sparse fourier representation via sampling. In: Proceedings of SPIE Wavelets XI, San Diego, CA 2005 (2005)
8. Gilbert, A.C., Strauss, M.J., Tropp, J.A., Vershynin, R.: One sketch for all: Fast algorithms for compressed sensing. In: 39th ACM Symposium on Theory of Computing (STOC'07)
9. Goldreich, O., Levin, L.: A hard-core predicate for all one-way functions. In: 27th ACM Symposium on Theory of Computing (STOC'89) (1989)
10. Mansour, Y.: Randomized interpolation and approximation of sparse polynomials. SIAM J. Comput. **24**, 357–368 (1995)
11. Sudan, M.: List decoding: algorithms and applications. SIGACT News **31**, 16–27 (2000)

LEDA: a Library of Efficient Algorithms
1995; Mehlhorn, Näher

CHRISTOS ZAROLIAGIS
Department of Computer Engineering and Informatics, University of Patras, Patras, Greece

Keywords and Synonyms

LEDA platform for combinatorial and geometric computing

Problem Definition

In the last forty years, there has been a tremendous progress in the field of computer algorithms, especially within the core area known as *combinatorial algorithms*. Combinatorial algorithms deal with objects such as lists, stacks, queues, sequences, dictionaries, trees, graphs, paths, points, segments, lines, convex hulls, etc, and constitute the basis for several application areas including network optimization, scheduling, transport optimization, CAD, VLSI design, and graphics. For over thirty years, asymptotic analysis has been the main model for designing and assessing the efficiency of combinatorial algorithms, leading to major algorithmic advances.

Despite so many breakthroughs, however, very little had been done (at least until 15 years ago) about the practical utility and assessment of this wealth of theoretical work. The main reason for this lack was the absence of a standard *algorithm library*, that is, of a software library that contains a systematic collection of robust and efficient implementations of algorithms and data structures, upon which other algorithms and data structures can be easily built.

The lack of an algorithm library limits severely the great impact which combinatorial algorithms can have.

The continuous re-implementation of basic algorithms and data structures slows down progress and typically discourages people to make the (additional) effort to use an efficient solution, especially if such a solution cannot be re-used. This makes the migration of scientific discoveries into practice a very slow process.

The major difficulty in building a library of combinatorial algorithms stems from the fact that such algorithms are based on complex data types, which are typically not encountered in programming languages (i. e., they are not built-in types). This is in sharp contrast with other computing areas such as statistics, numerical analysis, and linear programming.

Key Results

The currently most successful algorithm library is LEDA (Library for Efficient Data types and Algorithms) [4,5]. It contains a very large collection of advanced data structures and algorithms for combinatorial and geometric computing. The development of LEDA started in the early 1990s, it reached a very mature state in the late 1990s, and it continues to grow. LEDA has been written in C++ and has benefited considerably from the object-oriented paradigm.

Four major goals have been set in the design of LEDA.

1. *Ease of use*: LEDA provides a sizable collection of data types and algorithms in a form that they can be readily used by non-experts. It gives a precise and readable specification for each data type and algorithm, which is short, general and abstract (to hide the details of implementation). Most data types in LEDA are parameterized (e. g., the dictionary data type works for arbitrary key and information type). To access the objects of a data structure by position, LEDA has invented the *item concept* that casts positions into an abstract form.
2. *Extensibility*: LEDA is easily extensible by means of parametric polymorphism and can be used as a platform for further software development. Advanced data types are built on top of basic ones, which in turn rest on a uniform conceptual framework and solid implementation principles. The main mechanism to extend LEDA is through the so-called LEDA extension packages (LEPs). A LEP extends LEDA into a particular application domain and/or area of algorithms that is not covered by the core system. Currently, there are 15 such LEPs; for details see [1].
3. *Correctness*: In LEDA, programs should give sufficient justification (proof) for their answers to allow the user of a program to easily assess its correctness. Many algorithms in LEDA are accompanied by *program checkers*. A program checker C for a program P is a (typically very simple) program that takes as input the input of P, the output of P, and perhaps additional information provided by P, and verifies that the answer of P in indeed the correct one.
4. *Efficiency*: The implementations in LEDA are usually based on the asymptotically most efficient algorithms and data structures that are known for a problem. Quite often, these implementations have been fine-tuned and enhanced with heuristics that considerably improve running times. This makes LEDA not only the most comprehensive platform for combinatorial and geometric computing, but also a library that contains the currently fastest implementations.

Since 1995, LEDA is maintained by the Algorithmic Solutions Software GmbH [1] which is responsible for its distribution in academia and industry.

Other efforts for algorithm libraries include the Standard Template Library (STL) [7], the Boost Graph Library [2,6], and the Computational Geometry Algorithms Library (CGAL) [3].

STL [7] (introduced in 1994) is a library of interchangeable components for solving many fundamental problems on sequences of elements, which has been adopted into the C++ standard. It contributed the *iterator concept* which provides an interface between *containers* (an object that stores other objects) and algorithms. Each algorithm in STL is a function template parameterized by the types of iterators upon which it operates. Any iterator that satisfies a minimum set of requirements can be used regardless of the data structure accessed by the iterator. The systematic approach used in STL to build abstractions and interchangeable components is called *generic programming*.

The Boost Graph Library [2,6] is a C++ graph library that applies the notions of generic programming to the construction of graph algorithms. Each graph algorithm is written not in terms of a specific data structure, but instead in terms of a graph abstraction that can be easily implemented by many different data structures. This offers the programmer the flexibility to use graph algorithms in a wide variety of applications. The first release of the library became available in September 2000.

The Computational Geometry Algorithms Library [3] is another C++ library that focuses on geometric computing only. Its main goal is to provide easy access to efficient and reliable geometric algorithms to users in industry and academia. The CGAL library started in 1996 and the first release was in April 1998.

Among all libraries mentioned above LEDA is by far the best (both in quality and efficiency of implementations) regarding combinatorial computing. It is worth mentioning that the late versions of LEDA have also incorporated the iterator concept of STL.

Finally, a notable effort concerns the Stony Brook Algorithm Repository [8]. This is not an algorithm library, but a comprehensive collection of algorithm implementations for over seventy problems in combinatorial computing, started in 2001. The repository features implementations coded in different programming languages, including C, C++, Java, Fortran, ADA, Lisp, Mathematic, and Pascal.

Applications

An algorithm library for combinatorial and geometric computing has a wealth of applications in a wide variety of areas, including: network optimization, scheduling, transport optimization and control, VLSI design, computer graphics, scientific visualization, computer aided design and modeling, geographic information systems, text and string processing, text compression, cryptography, molecular biology, medical imaging, robotics and motion planning, and mesh partition and generation.

Open Problems

Algorithm libraries usually do not provide an interactive environment for developing and experimenting with algorithms. An important research direction is to add an interactive environment into algorithm libraries that would facilitate the development, debugging, visualization, and testing of algorithms.

Experimental Results

The are numerous experimental studies based on LEDA, STL, Boost, and CGAL, most of which can be found in the world-wide web. Also, the web sites of some of the libraries contain pointers to experimental work.

URL to Code

The afore mentioned algorithm libraries can be downloaded from their corresponding web sites, the details of which are given in the bibliography (Recommended Reading).

Cross References

► Engineering Algorithms for Large Network Applications
► Experimental Methods for Algorithm Analysis
► Implementation Challenge for Shortest Paths
► Shortest Paths Approaches for Timetable Information
► TSP-Based Curve Reconstruction

Recommended Reading

1. Algorithmic Solutions Software GmbH, http://www.algorithmic-solutions.com/. Accessed February 2008
2. Boost C++ Libraries, http://www.boost.org/. Accessed February 2008
3. CGAL: Computational Geometry Algorithms Library, http://www.cgal.org/. Accessed February 2008
4. Mehlhorn, K., Näher, S.: LEDA: A Platform for Combinatorial and Geometric Computing. Commun. ACM. **38**(1), 96–102 (1995)
5. Mehlhorn, K., Näher, S.. LEDA: A Platform for Combinatorial and Geometric Computing. Cambridge University Press, Boston (1999)
6. Siek, J., Lee, L.Q., Lumsdaine, A.: The Boost Graph Library. Addison-Wesley, Cambridge (2002)
7. Stepanov, A., Lee, M.: The Standard Template Library. In: Technical Report X3J16/94–0095, WG21/N0482, ISO Programming Language C++ Project. Hewlett-Packard, Palo Alto CA (1994)
8. The Stony Brook Algorithm Repository, http://www.cs.sunysb.edu/~algorith/. Accessed February 2008

Leontief Economy Equilibrium

2005; Codenotti, Saberi, Varadarajan, Ye
2005; Ye

YIN-YU YE
Department of Management Science and Engineering, Stanford University, Stanford, CA, USA

Keywords and Synonyms

Exchange market equilibrium with the leontief utility

Problem Definition

The Arrow–Debreu exchange market equilibrium problem was first formulated by Léon Walras in 1874 [7]. In this problem everyone in a population of m traders has an initial endowment of a divisible goods and a utility function for consuming all goods – their own and others'. Every trader sells the entire initial endowment and then uses the revenue to buy a bundle of goods such that his or her utility function is maximized. Walras asked whether prices could be set for everyone's goods such that this is possible. An answer was given by Arrow and Debreu in 1954 [1] who showed that, under mild conditions, such equilibrium would exist if the utility functions were concave. In general, it is unknown whether or not an equilibrium can be computed efficiently, see, e. g., ► General Equilibrium.

Consider a special class of Arrow–Debreu's problems, where each of the n traders has exactly one unit of a divisible and distinctive good for trade, and let trader i, $i = 1, \dots, n$, bring good i, which class of problems is called the *pairing class*. For given prices p_j on good j, consumer

i's maximization problem is

$$\begin{array}{ll} \text{maximize} & u_i(x_{i1}, \ldots, x_{in}) \\ \text{subject to} & \sum_j p_j x_{ij} \le p_i, \\ & x_{ij} \ge 0, \quad \forall j. \end{array} \tag{1}$$

Let x_i^* denote a maximal solution vector of (1). Then, vector p is called the Arrow–Debreu price equilibrium if there exists an x_i^* for consumer i, $i = 1, \ldots, n$, such that

$$\sum_i x_i^* = e$$

where e is the vector of all ones representing available goods on the exchange market.

The Leontief Economy Equilibrium problem is the Arrow–Debreu Equilibrium problem when the utility functions are in the Leontief form:

$$u_i(x_i) = \min_{j:\, h_{ij}>0} \left\{ \frac{x_{ij}}{h_{ij}} \right\},$$

where the Leontief coefficient matrix is given by

$$H = \begin{pmatrix} h_{11} & h_{12} & \ldots & h_{1n} \\ h_{21} & h_{22} & \ldots & h_{2n} \\ \ldots & \ldots & \ldots & \ldots \\ h_{n1} & h_{n2} & \ldots & h_{nn} \end{pmatrix}. \tag{2}$$

Here, one may assume that

Assumption 1 *H has no all-zero row, that is, every trader likes at least one good.*

Key Results

Let u_i be the equilibrium utility value of consumer i and p_i be the equilibrium price for good i, $i = 1, \ldots, n$. Also, let U and P are diagonal matrices whose diagonal entries are u_i's and p_i's, respectively. Then, the Leontief Economy Equilibrium $p \in R^n$, together with $u \in R^n$, must satisfy

$$\begin{array}{rcl} UHp & = & p, \\ P(e - H^{\mathrm{T}}u) & = & 0, \\ H^{\mathrm{T}}u & \le & e, \\ u,\ p & \ge & 0, \\ p & \ne & 0. \end{array} \tag{3}$$

One can prove:

Theorem 1 (Ye [8]) *Equation (3) always has a solution (u, p) under Assumption 1 (i. e., H has no all-zero row). However, a solution to Eq. (3) may not be a Leontief equilibrium, although every Leontief equilibrium satisfies Eq. (3).*

Theorem 2 (Ye [8]) *Let $B \subset \{1, 2, \ldots, n\}$, $N = \{1, 2, \ldots, n\} \setminus B$, H_{BB} be irreducible, and u_B satisfy the linear system*

$$H_{BB}^{\mathrm{T}} u_B = e, \quad H_{BN}^{\mathrm{T}} u_B \le e, \quad \text{and} \quad u_B > 0.$$

Then the (right) Perron–Frobenius eigen-vector p_B of $U_B H_{BB}$ together with $p_N = 0$ will be a solution to Eq. (3). And the converse is also true. Moreover, there is always a rational solution for every such B, that is, the entries of price vector are rational numbers, if the entries of H are rational. Furthermore, the size (bit-length) of the solution is bounded by the size (bit-length) of H.

The theorem implies that the traders in block B can trade among themselves and keep others goods "free". In particular, if one trader likes his or her own good more than any other good, that is, $h_{ii} \ge h_{ij}$ for all j, then $u_i = 1/h_{ii}$, $p_i = 1$, and $u_j = p_j = 0$ for all $j \ne i$, that is, $B = \{i\}$, makes a Leontief economy equilibrium. The theorem thus establishes, for the first time, a combinatorial algorithm to compute a Leontief economy equilibrium by finding a right block $B \ne \emptyset$, which is actually a non-trivial solution ($u \ne 0$) to an LCP problem

$$H^{\mathrm{T}}u + v = e,\ u^{\mathrm{T}}v = 0,\ 0 \ne u, v \ge 0. \tag{4}$$

If $H > 0$, then any complementary solution $u \ne 0$, together with its support $B = \{j\colon\ u_j > 0\}$, of Eq. (4) induce a Leontief economy equilibrium that is the (right) Perron–Frobenius eigen-vector of $U_B H_{BB}$, and it can be computed in polynomial time by solving a linear equation. Even if $H \not> 0$, any complementary solution $u \ne 0$ and $B = \{j\colon\ u_j > 0\}$, as long as H_{BB} is irreducible, induces an equilibrium for Eq. (3). The equivalence between the pairing Leontief economy model and the LCP also implies

Corollary 1 *LCP (4) always has a non-trivial solution, where H_{BB} is irreducible with $B = \{j\colon\ u_j > 0\}$, under Assumption 1 (i. e., H has no all-zero row).*

If Assumption 1 does not hold, the corollary may not be true; see example below:

$$H^{\mathrm{T}} = \begin{pmatrix} 0 & 2 \\ 0 & 1 \end{pmatrix}.$$

Applications

Given an arbitrary bimatrix game, specified by a pair of $n \times m$ matrices A and B, with positive entries, one can construct a Leontief exchange economy with $n + m$ traders and $n + m$ goods as follows. In words, trader i comes to the market with one unit of good i, for $i = 1, \ldots, n + m$.

Traders indexed by any $j \in \{1, \dots, n\}$ receive some utility only from goods $j \in \{n+1, \dots, n+m\}$, and this utility is specified by parameters corresponding to the entries of the matrix B. More precisely the proportions in which the j-th trader wants the goods are specified by the entries on the jth row of B. Vice versa, traders indexed by any $j \in \{n+1, \dots, n+m\}$ receive some utility only from goods $j \in \{1, \dots, n\}$. In this case, the proportions in which the j-th trader wants the goods are specified by the entries on the j-th column of A.

In the economy above, one can partition the traders in two groups, which bring to the market disjoint sets of goods, and are only interested in the goods brought by the group they do not belong to.

Theorem 3 (Codenotti et al. [4]) *Let (A,B) denote an arbitrary bimatrix game, where assume, w.l.o.g., that the entries of the matrices A and B are all positive. Let*

$$H^{\mathrm{T}} = \begin{pmatrix} 0 & A \\ B^{\mathrm{T}} & 0 \end{pmatrix}$$

describe the Leontief utility coefficient matrix of the traders in a Leontief economy. There is a one-to-one correspondence between the Nash equilibria of the game (A,B) and the market equilibria H of the Leontief economy. Furthermore, the correspondence has the property that a strategy is played with positive probability at a Nash equilibrium if and only if the good held by the corresponding trader has a positive price at the corresponding market equilibrium.

Gilboa and Zemel [6] proved a number of hardness results related to the computation of Nash equilibria (NE) for finite games in normal form. Since the NE for games with more than two players can be irrational, these results have been formulated in terms of NP-hardness for multiplayer games, while they can be expressed in terms of NP-completeness for two-player games. Using a reduction to the NE game, Codenotti et al. proved:

Theorem 4 (Codenotti et al. [4]) *It is NP-hard to decide whether a Leontief economy H has an equilibrium.*

Cross References

- ► Complexity of Bimatrix Nash Equilibria
- ► General Equilibrium
- ► Non-approximability of Bimatrix Nash Equilibria

Recommended Reading

The reader may want to read Brainard and Scarf [2] on how to compute equilibrium prices in 1891; Chen and Deng [3] on the most recent hardness result of computing the bimatrix game; Cottle et al. [5] for literature on linear complementarity problems; and all references listed in [4] and [8] for the recent literature on computational equilibrium.

1. Arrow, K.J., Debreu, G.: Existence of an equilibrium for competitive economy. Econometrica **22**, 265–290 (1954)
2. Brainard, W.C., Scarf, H.E.: How to compute equilibrium prices in 1891. Cowles Foundation Discussion Paper 1270, August 2000
3. Chen, X., Deng, X.: Settling the complexity of 2-player Nash-Equilibrium, ECCC TR05-140 (2005)
4. Codenotti, B., Saberi, A., Varadarajan, K., Ye, Y.: Leontief economies encode nonzero sum two-player games. SODA (2006)
5. Cottle, R., Pang, J.S., Stone, R.E.: The linear complementarity problem. Academic Press, Boston (1992)
6. Gilboa, I., Zemel, E.: Nash and correlated equilibria: some complexity considerations. Games Econ. Behav. **1**, 80–93 (1989)
7. Walras, L.: Elements of pure economics, or the theory of social wealth (1899, 4th ed; 1926, rev ed, 1954, Engl. Transl.) (1874)
8. Ye, Y.: Exchange market equilibria with leontief's utility: freedom of pricing leads to rationality. WINE (2005)

Linearity Testing/ Testing Hadamard Codes

1990; Blum, Luby, Rubinfeld

RONITT RUBINFELD
Department of Electrical Engineering and Computer Science, MIT, Cambridge, MA, USA

Keywords and Synonyms

Linearity testing; Testing Hadamard codes; Homomorphism testing

Problem Definition

This problem is concerned with distinguishing functions that are homomorphisms, i. e. satisfying $\forall x, y, f(x) + f(y) = f(x+y)$, from those functions that must be changed on at least ϵ fraction of the domain in order to be turned into a homomorphism, given query access to the function. This problem was initially motivated by applications to testing programs which compute linear functions [8]. Since Hadamard codes are such that the codewords are exactly the evaluations of linear functions over boolean variables, a solution to this problem gives a way of distinguishing codewords of the Hadamard code from those strings that are far in relative Hamming distance from codewords. These algorithms were in turn used in the constructions of Probabilistically Checkable Proof Systems (cf. [3]). Further work has extended these techniques to testing other properties of low degree polynomials and solutions to other addition theorems [3,9,24,25].

Notations

For two finite groups G, H (not necessarily Abelian), an arbitrary map $f : G \to H$ is a *homomorphism* if

$$\forall x, y,\ f(x) \times f(y) = f(x \times y)\ .$$

f is *ϵ-close to a homomorphism* if there is some homomorphism g such that g and f differ on at most $\epsilon|G|$ elements of G, and f is *ϵ-far* otherwise.

Given a parameter $0 \le \epsilon \le 1$, and query access to a function $f\colon G \to H$, a homomorphism tester (also refeered to as a *linearity tester* in the literature) is an algorithm $\mathcal{T}$ which outputs "Pass" if f is a homomorphism, and "Fail" if f is ϵ-far from a homomorphism. The homomorphism tester should err with probability at most 1/3 for any f.[1]

For two finite groups G, H (not necessarily Abelian), an arbitrary map $f : G \to H$, and a parameter $0 < \epsilon < 1$, define δ_f (the subscript is dropped and this is referred to as δ, when f is obvious from the context) then the *probability of group law failure*, by

$$1 - \delta = \Pr_{x,y}\left[f(x) \times f(y) = f(x \times y)\right]\ .$$

Define τ such that τ is the minimum ϵ for which f is ϵ-close to a homomorphism.

Problem 1 *For f, δ as above, is it possible to upper bound τ in terms of a function that depends on the probability of group law failure δ, but not on the size of the domain $|G|$?*

Key Results

Blum, Luby and Rubinfeld [8], considered this question and showed that over cyclic groups, there is a constant δ_0, such that if $\delta \le \delta_0$, then the one can upper bound ϵ in terms of a function of δ that is independent of $|G|$. This yields a homomorphism tester with query complexity that depends (polynomially) on $1/\epsilon$, but is independent of $|G|$. The final version of [8] contains an improved argument due to Coppersmith [10], which applies to all Abelian groups, shows that $\delta_0 < 2/9$ suffices, and that ϵ is upper bounded by the smaller root of $x(1-x) = \delta$ (yielding a homomorphism tester with query complexity linear in $1/\epsilon$). Furthermore, the bound on δ_0 was shown to be tight for general groups [10].

In [6], a relationship between the probability of group law failure and the closeness to being a homomorphism was established that applies to general (non-Abelian) groups. For a given δ, let $\tau = (3 - \sqrt{9 - 24\delta})/12 \le \delta/2$ be the smaller root of $3x - 6x^2 = \delta$. In [6] it is shown that for $\delta_0 < 2/9$, then f is τ-close to a homomorphism. The condition on δ, and the bound on τ as a function of δ, are shown to be tight. The latter improves on the relationship given in [8,10].

There has been interest in improving various parameters of homomorphism testing results, due to their use in the construction of Probabilistically Checkable Proof Systems (cf. [3]). In particular, both the constant δ_0 and the number of random bits required by the homomorphism test affect the efficiency of the proof system and in turn the hardness of approximation results that one can achieve using the proof system.

The homomorphism testing results can be improved in some cases: It has been previously mentioned that $\delta_0 < 2/9$ is optimal over general Abelian groups [10]. However, using Fourier techniques, Bellare et al. [5] have shown that for groups of the form $(\mathbf{Z}/2)^n$, $\delta_0 \le 45/128$ suffices. For such δ_0, $\epsilon < \delta$. Kiwi later provided a similar result based on the discrete Fourier transform and weight distributions to improve the bound on the dependence of ϵ on δ [17].

Several works have shown methods of reducing the number of random bits required by the homomorphism tests. That is, in the natural implementation of the homomorphism test, $2\log|G|$ random bits per trial are used to pick x,y. The results of [7,14,26,28,29] have shown that fewer random bits are sufficient for implementing the homomorphism tests. In particular, Trevisan [29] and Samorodnitsky and Trevisan [26] have considered the "amortized query complexity" of testing homomorphisms, which is a measure that quantifies the trade-off between the query complexity of the testing algorithm and the probability of accepting the function. Homomorphism tests with low amortized query complexity are useful in constructing PCP systems with low amortized query complexity. A simpler analysis which improves the dependence of the acceptance probability in terms of the distance of the tested function to the closest linear function is given in [14]. The work of [28] gives a homomorphism test for general (non-Abelian) groups that uses only $(1 + o(1))\log_2|G|$ random bits. Given a Cayley graph that is an expander with normalized second eigenvalue γ, and

[1]The choice of 1/3 is arbitrary. Using standard techniques, any homomorphism tester satisfying 1/3 error probability can be turned into a homomorphism tester with $0 < \beta < 1/3$ error probability by repeating the original tester $O(\log \frac{1}{\beta})$ times and taking the majority answer.

for the analogous definitions of δ, τ, they show that for $\delta < (1-\gamma)/12$, τ is upper bounded by $4\delta/(1-\gamma)$. Very recently, Samordnitsky and Trevisan [27] have considered a relaxed version of a homomorphism test which accepts linear functions and rejects functions with low influences.

The case when G is a subset of an infinite group, f is a real-valued function and the oracle query to f returns a finite precision approximation to $f(x)$ has been considered in [2,11,12,20,21], and testers with query complexity that are independent of the domain size have been given (see [19] for a survey).

A Related Problem on Convolutions of Distributions

In the following, a seemingly unrelated question about distributions that are close to their self-convolutions is mentioned: Let $A = \{a_g | g \in G\}$ be a distribution on group G. The convolution of distributions A, B is

$$C = A * B, \;\; c_x = \sum_{y,z\in G;\; yz=x} a_y b_z \; .$$

Let A' be the *self-convolution* of A, $A * A$, i. e. $a'_x = \sum_{y,z\in G; yz=x} a_y a_z$. It is known that $A = A'$ exactly when A is the uniform distribution over a subgroup of G. Suppose it is known that A is close to A', can one say anything about A in this case? Suppose $dist(A, A') = \frac{1}{2}\sum_{x\in G}|a_x - a'_x| \le \epsilon$ for small enough ϵ. Then [6] show that A must be close to the uniform distribution over a subgroup of G. More precisely, in [6] it is shown that for a distribution A over a group G, if $dist(A, A') = \frac{1}{2}\sum_{x\in G}|a_x - a'_x| \le \epsilon \le 0.0273$, then there is a subgroup H of G such that $dist(A, U_H) \le 5\epsilon$, where U_H is the uniform distribution over H [6]. On the other hand, in [6] there is an example of a distribution A such that $dist(A, A * A) \approx .1504$, but A is not close to uniform on any subgroup of the domain.

A weaker version of this result, was used to prove a preliminary version of the homomorphism testing result in [8]. To give a hint of why one might consider the question on convolutions of distributions when investigating homomorphism testing, consider the distribution A_f achieved by picking x uniformly from G and outputting $f(x)$. It is easy to see that the error probability δ in the homomorphism test is at least $dist(A_f, A_f * A_f)$. The other, more useful, direction is less obvious. In [6] it is shown that this question on distributions is "equivalent" in difficulty to homomorphism testing:

Theorem 1 *Let G, H be finite groups. Assume that there is a parameter β_0 and function ϕ such that the following holds:*

For all distributions A over group G, if $dist(A * A, A) \le \beta \le \beta_0$ then A is $\phi(\beta)$-close to uniform over a subgroup of G.

Then, for any $f : G \to H$ and $\delta < \beta_0$ such that $1 - \delta = Pr[f(x) * f(y) = f(x * y)]$, and $\phi(\delta) \le 1/2$, it is the case that f is $\phi(\delta)$-close to a homomorphism.

Applications

Self-Testing/Correcting Programs

When a program has not been verified and therefore is not known to be correct on all inputs (or possibly even known to be incorrect on some inputs), [8] have suggested the following approach: take programs that are known to be correct on most inputs and apply a simple transformation to produce a program that is correct on every input. This transformation is referred to as producing a *self-corrector*. Moreover, for many functions, one can actually test that the program for f is correct on most inputs, without the aid of another program for f that has already been verified. Such testers for programs are referred to as *self-testers*.

The homomorphism testing problem was initially motivated by applications to constructing self-testers for programs which purport to compute various linear functions [8]. Such functions include integer, polynomial, matrix and modular multiplication and division. Once it is verified that a program agrees on most inputs with a specific linear function, the task of determining whether it agrees with the *correct* linear function on most inputs becomes much easier.

Furthermore, for programs purporting to compute linear functions, it is very simple to construct self-correctors: Assume one is given a program which on input x outputs $f(x)$, such that f agrees on most inputs with linear function g. Consider the algorithm that picks $c \log 1/\beta$ values y, computes $f(x + y) - f(y)$ and outputs the value that is seen most often. If f is $\frac{1}{8}$-close to g, then since both y and $x + y$ are uniformly distributed, it is the case that for at least 3/4 of the choices of y, $g(x + y) = f(x + y)$ *and* $g(y) = f(y)$, in which case $f(x + y) - f(y) = g(x)$. Thus it is easy to show that there is a constant c such that if f is $\frac{1}{8}$-close to a homomorphism g, then for all x, the above algorithm will output $g(x)$ with probability at least $1 - \beta$.

Probabilistically Checkable Proofs

An equivalent formulation of the homomorphism testing problem is in terms of the query complexity of testing a codeword of a Hadamard code. The results mentioned about have been used to construct Probabilistically Checkable Proof systems which can be verified with very few queries (cf. [3,13]).

Open Problems

It is natural to wonder what other classes of functions have testers whose efficiency is sublinear in the domain size? There are some partial answers to this question: The field of functional equations is concerned with the prototypical problem of characterizing the set of functions that satisfy a given set of properties (or functional equations). For example, the class of functions of the form $f(x) = \tan Ax$ are characterized by the functional equation

$$\forall x, y, f(x+y) = \frac{f(x)+f(y)}{1-f(x)f(y)} .$$

D'Alembert's equation

$$\forall x, y, f(x+y) + f(x-y) = 2f(x)f(y)$$

characterizes the functions $0, \cos Ax, \cosh Ax$. Multivariate polynomials of total degree d over Z_p for $p > md$ can be characterized by the equation

$$\forall \hat{x}, \hat{h} \in Z_p^m, \sum_{i=0}^{d+1} \alpha_i f(\hat{x} + i\hat{h}) = 0 ,$$

where $\alpha_i = (-1)^{i+1}\binom{d+1}{i}$. All of the above characterizations are known to yield testers for the corresponding function families whose query complexity is independent of the domain size (though for the case of polynomials, there is a polynomial dependence on the total degree d) [9,24,25]. A long series of works have given increasingly efficient to test characterizations of functions that are low total degree polynomials (cf. [1,3,4,15,18,22,23]).

A second line of questions that remain to be understood regard which codes are such that strings can be quickly tested to determine whether they are close to a codeword? Some initial work on this questions is given in [1,15,16,18].

Cross References

► Learning Heavy Fourier Coefficients of Boolean Functions

Recommended Reading

1. Alon, N., Kaufman, T., Krivilevich, M., Litsyn, S., Ron, D.: Testing low-degree polynomials over gf(2). In: Proceedings of RANDOM '03. Lecture Notes in Computer Science, vol. 2764, pp. 188–199. Springer, Berlin Heidelberg (2003)
2. Ar, S., Blum, M., Codenotti, B., Gemmell, P.: Checking approximate computations over the reals. In: Proceedings of the Twenty-Fifth Annual ACM Symposium on the Theory of Computing, pp. 786–795. ACM, New York (2003)
3. Arora, S., Lund, C., Motwani, R., Sudan, M., Szegedy, M.: Proof verification and the hardness of approximation problems. J. ACM **45**(3), 501–555 (1998)
4. Arora, S., Sudan, M.: Improved low degree testing and its applications. In: Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, pp. 485–495. ACM, New York (1997)
5. Bellare, M., Coppersmith, D., Håstad, J., Kiwi, M., Sudan, M.: Linearity testing over characteristic two. IEEE Trans. Inf. Theory **42**(6), 1781–1795 (1996)
6. Ben-Or, M., Coppersmith, D., Luby, M., Rubinfeld, R.: Non-abelian homomorphism testing, and distributions close to their self-convolutions. In: Proceedings of APPROX-RANDOM. Lecture Notes in Computer Science, vol. 3122, pp. 273–285. Springer, Berlin Heidelberg (2004)
7. Ben-Sasson, E., Sudan, M., Vadhan, S., Wigderson, A.: Randomness-efficient low degree tests and short pcps via epsilon-biased sets. In: Proceedings of the Thirty-Fifth Annual ACM Symposium on the Theory of Computing, pp. 612–621. ACM, New York (2003)
8. Blum, M., Luby, M., Rubinfeld, R.: Self-testing/correcting with applications to numerical problems. J. CSS **47**, 549–595 (1993)
9. Cleve, R., Luby, M.: A note on self-testing/correcting methods for trigonometric functions. In: International Computer Science Institute Technical Report TR-90-032, July 1990
10. Coppersmith, D.: Manuscript, private communications (1989)
11. Ergun, F., Kumar, R., Rubinfeld, R.: Checking approximate computations of polynomials and functional equations. SIAM J. Comput. **31**(2), 550–576 (2001)
12. Gemmell, P., Lipton, R., Rubinfeld, R., Sudan, M., Wigderson, A.: Self-testing/correcting for polynomials and for approximate functions. In: Proceedings of the Twenty-Third Annual ACM Symposium on Theory of Computing, pp. 32–42. ACM, New York (1991)
13. Hastad, J.: Some optimal inapproximability results. J. ACM **48**(4), 798–859 (2001)
14. Hastad, J., Wigderson, A.: Simple analysis of graph tests for linearity and pcp. Random Struct. Algorithms **22**(2), 139–160 (2003)
15. Jutla, C., Patthak, A., Rudra, A., Zuckerman, D.: Testing low-degree polynomials over prime fields. In: Proceedings of the Forty-Fifth Annual Symposium on Foundations of Computer Science, pp. 423–432. IEEE, New York (2004)
16. Kaufman, T., Litsyn, S.: Almost orthogonal linear codes are locally testable. In: Proceedings of the Forty-Sixth Annual Symposium on Foundations of Computer Science, pp. 317–326. IEEE, New York (2005)
17. Kaufman, T., Litsyn, S., Xie, N.: Breaking the ϵ-soundness bound of the linearity test over gf(2). Electronic Colloquium on Computational Complexity, Report TR07–098, October 2007
18. Kaufman, T., Ron, D.: Testing polynomials over general fields. In: Proceedings of the Forty-Fifth Annual Symposium on Foundations of Computer Science, pp. 413–422. IEEE, New York (2004)
19. Kiwi, M., Magniez, F., Santha, M.: Exact and approximate testing/correcting of algebraic functions: A survey. Theoretical Aspects Compututer Science, LNCS **2292**, 30–83 (2001)

20. Kiwi, M., Magniez, F., Santha, M.: Approximate testing with error relative to input size. J. CSS **66**(2), 371–392 (2003)
21. Magniez, F.: Multi-linearity self-testing with relative error. Theory Comput. Syst. **38**(5), 573–591 (2005)
22. Polischuk, A., Spielman, D.: Nearly linear-size holographic proofs. In: Proceedings of the Twenty-Sixth Annual ACM Symposium on the Theory of Computing, pp. 194–203. ACM, New York (1994)
23. Raz, R., Safra, S.: A sub-constant error-probability low-degree test, and a sub-constant error-probability pcp characterization of np. In: Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, pp. 475–484. ACM, New York (1997)
24. Rubinfeld, R.: On the robustness of functional equations. SIAM J. Comput. **28**(6), 1972–1997 (1999)
25. Rubinfeld, R., Sudan, M.: Robust characterization of polynomials with applications to program testing. SIAM J. Comput. **25**(2), 252–271 (1996)
26. Samorodnitsky, A., Trevisan, L.: A PCP characterization of NP with optimal amortized query complexity. In: Proceedings of the Thirty-Second Annual ACM Symposium on the Theory of Computing, pp. 191–199. ACM, New York (2000)
27. Samorodnitsky, A., Trevisan, L.: Gowers uniformity, influence of variables, and pcps. In: Thirty-Eighth ACM Symposium on Theory of Computing, pp. 11–20. ACM, New York (2006)
28. Shpilka, A., Wigderson, A.: Derandomizing homomorphism testing in general groups. In: Proceedings of the Thirty-Sixth Annual ACM Symposium on the Theory of Computing, pp. 427–435. ACM, NY, USA (2004)
29. Trevisan, L.: Recycling queries in pcps and in linearity tests. In: Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, pp. 299–308. ACM, New York (1998)

Linearizability

1990; Herlihy, Wing

MAURICE HERLIHY
Department of Computer Science, Brown University, Providence, RI, USA

Keywords and Synonyms

Atomicity

Problem Definition

An *object* in languages such as Java and C++ is a container for data. Each object provides a set of *methods* that are the only way to to manipulate that object's internal state. Each object has a *class* which defines the methods it provides and what they do.

In the absence of concurrency, methods can be described by a pair consisting of a *precondition* (describing the object's state before invoking the method) and a *postcondition*, describing, once the method returns, the object's state and the method's return value. If, however, an object is shared by concurrent threads in a multiprocessor system, then method calls may overlap in time, and it no longer makes sense to characterize methods in terms of pre- and post-conditions.

Linearizability is a correctness condition for concurrent objects that characterizes an object's concurrent behavior in terms of an "equivalent" sequential behavior. Informally, the object behaves "as if" each method call takes effect instantaneously at some point between its invocation and its response. This notion of correctness has some useful formal properties. First, it is *non-blocking*, which means that linearizability as such never requires one thread to wait for another to complete an ongoing method call. Second, it is *local*, which means that an object composed of linearizable objects is itself linearizable. Other proposed correctness conditions in the literature lack at least one of these properties.

Notation

An execution of a concurrent system is modeled by a *history*, a finite sequence of method *invocation* and *response events*. A *subhistory* of a history H is a subsequence of the events of H. A method invocation is written as $\langle x.m(a^*)A \rangle$, where x is an object, m a method name, a^* a sequence of arguments, and A a thread. A method response is written as $\langle x : t(r^*)A \rangle$ where t is a termination condition and r^* is a sequence of result values.

A response *matches* an invocation if their objects and thread names agree. A *method call* is a pair consisting of an invocation and the next matching response. An invocation is *pending* in a history if no matching response follows the invocation. If H is a history, *complete*(H) is the subsequence of H consisting of all matching invocations and responses. A history H is *sequential* if the first event of H is an invocation, and each invocation, except possibly the last, is immediately followed by a matching response.

Let H be a a history. The *thread subhistory* $H|P$ is the subsequence of events in H with thread name P. The *object subhistory* $H|x$ is similarly defined for an object x. Two histories H and H' are *equivalent* if for every thread A, $H|A = H'|A$. A history H is *well-formed* if each thread subhistory $H|A$ of H is sequential. Notice that thread subhistories of a well-formed history are always sequential, but object subhistories need not be.

A *sequential specification* for an object is a prefix-closed set of sequential object histories that defines that object's *legal* histories. A sequential history H is *legal* if each object subhistory is legal. A method is *total* if it is defined for every object state, otherwise it is *partial*. (For example, a *deq()* method that blocks on an empty queue is partial, while one that throws an exception is total.)

A history H defines an (irreflexive) partial order $\rightarrow_H$ on its method calls: $m_0 \rightarrow_H m_1$ if the result event of m_0 occurs before the invocation event of m_1. If H is a sequential history, then $\rightarrow_H$ is a total order.

Let H be a history and x an object such that $H|x$ contains method calls m_0 and m_1. A call $m_0 \rightarrow_x m_1$ if m_0 precedes m_1 in $H|x$. Note that $\rightarrow_x$ is a total order.

Informally, linearizability requires that each method call appear to "take effect" instantaneously at some moment between its invocation and response. An important implication of this definition is that method calls that do not overlap cannot be reordered: linearizability preserves the "real-time" order of method calls. Formally,

Definition 1 A history H is *linearizable* if it can be extended (by appending zero or more response events) to a history H' such that:

- L1 *complete*(H') is equivalent to a legal sequential history S, and
- L2 If method call m_0 precedes method call m_1 in H, then the same is true in S.

S is called a *linearization* of H. (H may have multiple linearizations.) Informally, extending H to H' captures the idea that some pending invocations may have taken effect even though their responses have not yet been returned to the caller.

Key Results

The Locality Property

A property is *local* if all objects collectively satisfy that property provided that each individual object satisfies it.

Linearizability is local:

Theorem 1 *H is linearizable if and only if $H|x$ is linearizable for ever object x.*

Proof The "only if" part is obvious.

For each object x, pick a linearization of $H|x$. Let R_x be the set of responses appended to $H|x$ to construct that linearization, and let $\rightarrow_x$ be the corresponding linearization order. Let H' be the history constructed by appending to H each response in R_x.

The $\rightarrow_H$ and $\rightarrow_x$ orders can be "rolled up" into a single partial order. Define the relation $\rightarrow$ on method calls of $complete(H')$: For method calls m and $\bar{m}$, $m \rightarrow \bar{m}$ if there exist method calls $m_0, \ldots, m_n$, such that $m = m_0, \bar{m} = m_n$, and for each i between 0 and $n-1$, either $m_i \rightarrow_x m_{i+1}$ for some object x, or $m_i \rightarrow_H m_{i+1}$.

It turns out that $\rightarrow$ is a partial order. Clearly, $\rightarrow$ is transitive. It remains to be shown that $\rightarrow$ is anti-reflexive: for all x, it is false that $x \rightarrow x$.

The proof proceeds by contradiction. If not, then there exist method calls $m_0, \ldots, m_n$, such that $m_0 \rightarrow m_1 \rightarrow \cdots \rightarrow m_n, m_n \rightarrow m_0$, and each pair is directly related by some $\rightarrow_x$ or by $\rightarrow_H$.

Choose a cycle whose length is minimal. Suppose all method calls are associated with the same object x. Since $\rightarrow_x$ is a total order, there must exist two method calls m_{i-1} and m_i such that $m_{i-1} \rightarrow_H m_i$ and $m_i \rightarrow_x m_{i-1}$, contradicting the linearizability of x.

The cycle must therefore include method calls of at least two objects. By reindexing if necessary, let m_1 and m_2 be method calls of distinct objects. Let x be the object associated with m_1. None of $m_2, \ldots, m_n$ can be a method call of x. The claim holds for m_2 by construction. Let m_i be the first method call in $m_3, \ldots, m_n$ associated with x. Since m_{i-1} and m_i are unrelated by $\rightarrow_x$, they must be related by $\rightarrow_H$, so the response of m_{i-1} precedes the invocation of m_i. The invocation of m_2 precedes the response of m_{i-1}, since otherwise $m_{i-1} \rightarrow_H m_2$, yielding the shorter cycle $m_2, \ldots, m_{i-1}$. Finally, the response of m_1 precedes the invocation of m_2, since $m_1 \rightarrow_H m_2$ by construction. It follows that the response to m_1 precedes the invocation of m_i, hence $m_1 \rightarrow_H m_i$, yielding the shorter cycle $m_1, m_i, \ldots, m_n$.

Since m_n is not a method call of x, but $m_n \rightarrow m_1$, it follows that $m_n \rightarrow_H m_1$. But $m_1 \rightarrow_H m_2$ by construction, and because $\rightarrow_H$ is transitive, $m_n \rightarrow_H m_2$, yielding the shorter cycle $m_2, \ldots, m_n$, the final contradiction. ☐

Locality is important because it allows concurrent systems to be designed and constructed in a modular fashion; linearizable objects can be implemented, verified, and executed independently. A concurrent system based on a non-local correctness property must either rely on a centralized scheduler for all objects, or else satisfy additional constraints placed on objects to ensure that they follow compatible scheduling protocols. Locality should not be taken for granted; as discussed below, the literature includes proposals for alternative correctness properties that are not local.

The Non-Blocking Property

Linearizability is a *non-blocking* property: a pending invocation of a total method is never required to wait for another pending invocation to complete.

Theorem 2 *Let inv(m) be an invocation of a total method. If $\langle x\ invP\rangle$ is a pending invocation in a linearizable history H, then there exists a response $\langle x\, resP\rangle$ such that $H\cdot\langle x\, resP\rangle$ is linearizable.*

Proof Let S be any linearization of H. If S includes a response $\langle x\ resP \rangle$ to $\langle x\ invP \rangle$, the proof is complete, since S is also a linearization of $H \cdot \langle x\ resP \rangle$. Otherwise, $\langle x\ invP \rangle$ does not appear in S either, since linearizations, by definition, include no pending invocations. Because the method is total, there exists a response $\langle x\ resP \rangle$ such that

$$S' = S \cdot \langle x\ invP \rangle \cdot \langle x\ res\ P \rangle$$

is legal. S', however, is a linearization of $H \cdot \langle x\ resP \rangle$, and hence is also a linearization of H. □

This theorem implies that linearizability by itself never forces a thread with a pending invocation of a total method to block. Of course, blocking (or even deadlock) may occur as artifacts of particular implementations of linearizability, but it is not inherent to the correctness property itself. This theorem suggests that linearizability is an appropriate correctness condition for systems where concurrency and real-time response are important. Alternative correctness conditions, such as serializability [1] do not share this non-blocking property.

The non-blocking property does not rule out blocking in situations where it is explicitly intended. For example, it may be sensible for a thread attempting to dequeue from an empty queue to block, waiting until another thread enqueues an item. The queue specification captures this intention by making the `deq()` method's specification partial, leaving it's effect undefined when applied to an empty queue. The most natural concurrent interpretation of a partial sequential specification is simply to wait until the object reaches a state in which the method is defined.

Other Correctness Properties

Sequential Consistency [4] is a weaker correctness condition that requires Property *L1* but not *L2*: method calls must appear to happen in some one-at-a-time, sequential order, but calls that do not overlap can be reordered. Every linearizable history is sequentially consistent, but not vice versa. Sequential consistency permits more concurrency, but it is not a local property: a system composed of multiple sequentially-consistent objects is not itself necessarily sequentially consistent.

Much work on databases and distributed systems uses *serializability* as the basic correctness condition for concurrent computations. In this model, a *transaction* is a "thread of control" that applies a finite sequence of methods to a set of objects shared with other transactions. A history is *serializable* if it is equivalent to one in which transactions appear to execute sequentially, that is, without interleaving. A history is *strictly serializable* if the transactions' order in the sequential history is compatible with their precedence order: if every method call of one transaction precedes every method call of another, the former is serialized first. (Linearizability can be viewed as a special case of strict serializability where transactions are restricted to consist of a single method applied to a single object.)

Neither serializability nor strict serializability is a local property. If different objects serialize transactions in different orders, then there may be no serialization order common to all objects. Serializability and strict serializability are *blocking* properties: Under certain circumstances, a transaction may be unable to complete a pending method without violating serializability. A *deadlock* results if multiple transactions block one another. Such transactions must be rolled back and restarted, implying that additional mechanisms must be provided for that purpose.

Applications

Linearizability is widely used as the basic correctness condition for many concurrent data structure algorithms [5], particularly for lock-free and wait-free data structures [2]. Sequential consistency is widely used for describing low-level systems such as hardware memory interfaces. Serializability and strict serializability are widely used for database systems in which it must be easy for application programmers to preserve complex application-specific invariants spanning multiple objects.

Open Problems

Modern multiprocessors often support very weak models of memory consistency. There are many open problems concerning how to model such behavior, and how to ensure linearizable object implementations on top of such architectures.

Cross References

► Concurrent Programming, Mutual Exclusion
► Registers

Recommended Reading

The notion of Linearizability is due to Herlihy and Wing [3], while Sequential Consistency is due to Lamport [4], and serializability to Eswaran et al. [1].

1. Eswaran, K.P., Gray, J.N., Lorie, R.A., Traiger, I.L.: The notions of consistency and predicate locks in a database system. Commun. ACM **19**(11), 624–633 (1976). doi: http://doi.acm.org/10.1145/360363.360369

2. Herlihy, M.: Wait-free synchronization. ACM Trans. Program. Lang. Syst. (TOPLAS) **13**(1), 124–149 (1991)
3. Herlihy, M.P., Wing, J.M.: Linearizability: a correctness condition for concurrent objects. ACM Trans. Program. Lang. Syst. (TOPLAS) **12**(3), 463–492 (1990)
4. Lamport, L.: How to make a multiprocessor computer that correctly executes multiprocess programs. IEEE Trans. Comput. **C-28**(9), 690 (1979)
5. Vafeiadis, V., Herlihy, M., Hoare, T., Shapiro, M.: Proving correctness of highly-concurrent linearisable objects. In: PPoPP '06: Proceedings of the eleventh ACM SIGPLAN symposium on Principles and practice of parallel programming, pp. 129–136 (2006). doi: http://doi.acm.org/10.1145/1122971.1122992

List Decoding near Capacity: Folded RS Codes
2006; Guruswami, Rudra

ATRI RUDRA
Department of Computer Science and Engineering, University at Buffalo, State University of New York, Buffalo, NY, USA

Keywords and Synonyms

Decoding; Error correction

Problem Definition

One of the central trade-offs in the theory of error-correcting codes is the one between the amount of redundancy needed and the fraction of errors that can be corrected.[1] The redundancy is measured by the *rate* of the code, which is the ratio of the the number of information symbols in the message to that in the codeword – thus, for a code with encoding function $E : \Sigma^k \to \Sigma^n$, the rate equals k / n. The *block length* of the code equals n, and Σ is its *alphabet*.

The goal in *decoding* is to find, given a noisy received word, the actual codeword that it could have possibly resulted from. If the target is to correct a fraction ρ of errors (ρ will be called the error-correction radius), then this amounts to finding codewords within (normalized Hamming) distance ρ from the received word. We are guaranteed that there will be a unique such codeword provided the distance between *every* two distinct codewords is at least 2ρ, or in other words the relative distance of the code is at least 2ρ. However, since the relative distance δ of a code must satisfy $\delta \le 1 - R$ where R is the rate of the code (by the Singleton bound), if one insists on an unique answer, the best trade-off between ρ and R is $\rho = \rho_U(R) = (1 - R)/2$. But this is an overly pessimistic estimate of the error-correction radius, since the way Hamming spheres pack in space, for *most* choices of the received word there will be at most one codeword within distance ρ from it even for ρ much greater than $\delta/2$. Therefore, *always* insisting on a unique answer will preclude decoding most such received words owing to a few pathological received words that have more than one codeword within distance roughly $\delta/2$ from them.

[1]This entry deals with the adversarial or worst-case model of errors—no assumption is made on how the errors and error locations are distributed beyond an upper bound on the total number of errors that may be caused.

A notion called list decoding, that dates back to the late 1950's [1,9], provides a clean way to get around this predicament, and yet deal with worst-case error patterns. Under list decoding, the decoder is required to output a list of all codewords within distance ρ from the received word. Let us call a code C (ρ, L)-*list decodable* if the number of codewords within distance ρ of any received word is at most L. To obtain better trade-offs via list decoding, (ρ, L)-list decodable codes are needed where L is bounded by a polynomial function of the block length, since this an *a priori* requirement for polynomial time list decoding. How large can ρ be as a function of R for which such (ρ, L)-list decodable codes exist? A standard random coding argument shows that $\rho \ge 1 - R - o(1)$ can be achieved over large enough alphabets, cf. [2,10], and a simple counting argument shows that ρ must be at most $1 - R$. Therefore the *list decoding capacity*, i. e., the information-theoretic limit of list decodability, is given by the trade-off $\rho_{\text{cap}}(R) = 1 - R = 2\rho_U(R)$. Thus list decoding holds the promise of correcting *twice* as many errors as unique decoding, for *every* rate. The above-mentioned list decodable codes are non-constructive. In order to realize the potential of list decoding, one needs explicit constructions of such codes, and on top of that, polynomial time algorithms to perform list decoding.

Building on works of Sudan [8], Guruswami and Sudan [6] and Parvaresh and Vardy [7], Guruswami and Rudra [5] present codes that get arbitrarily close to the list decoding capacity $\rho_{\text{cap}}(R)$ for every rate. In particular, for every $1 > R > 0$ and every $\epsilon > 0$, they give *explicit* codes of rate R together with polynomial time list decoding algorithm that can correct up to a fraction $1 - R - \epsilon$ of errors. These are the first explicit codes (with efficient list decoding algorithms) that get arbitrarily close to the list decoding capacity for *any* rate.

Description of the Code

Consider a Reed–Solomon (RS) code $C = \mathrm{RS}_{\mathbb{F},\mathbb{F}^*}[n, k]$ consisting of evaluations of degree k polynomials over some finite field $\mathbb{F}$ at the set $\mathbb{F}^*$ of nonzero elements of $\mathbb{F}$. Let $q = |\mathbb{F}| = n + 1$. Let γ be a generator of the multiplicative group $\mathbb{F}^*$, and let the evaluation points be ordered as $1, \gamma, \gamma^2, \dots, \gamma^{n-1}$. Using all nonzero field elements as evaluation points is one of the most commonly used instantiations of Reed–Solomon codes.

Let $m \geq 1$ be an integer parameter called the *folding parameter*. For ease of presentation, it will assumed that m divides $n = q - 1$.

Definition 1 (Folded Reed–Solomon Code) The m-folded version of the RS code C, denoted $\mathsf{FRS}_{\mathbb{F},\gamma,m,k}$, is a code of block length $N = n/m$ over $\mathbb{F}^m$. The encoding of a message $f(X)$, a polynomial over $\mathbb{F}$ of degree at most k, has as its j'th symbol, for $0 \leq j < n/m$, the m-tuple $(f(\gamma^{jm}), f(\gamma^{jm+1}), \cdots, f(\gamma^{jm+m-1}))$. In other words, the codewords of $C' = \mathsf{FRS}_{\mathbb{F},\gamma,m,k}$ are in one-one correspondence with those of the RS code C and are obtained by bundling together consecutive m-tuple of symbols in codewords of C.

Key Results

The following is the main result of Guruswami and Rudra.

Theorem 1 ([5]) *For every $\epsilon > 0$ and $0 < R < 1$, there is a family of folded Reed–Solomon codes that have rate at least R and which can be list decoded up to a fraction $1 - R - \epsilon$ of errors in time (and outputs a list of size at most) $(N/\epsilon^2)^{O(\epsilon^{-1}\log(1/R))}$ where N is the block length of the code. The alphabet size of the code as a function of the block length N is $(N/\epsilon^2)^{O(1/\epsilon^2)}$.*

The result of Guruswami and Rudra also works in a more general setting called *list recovering*, which is defined next.

Definition 2 (List Recovering) A code $C \subseteq \Sigma^n$ is said to be (ζ, l, L)-list recoverable if for every sequence of sets $S_1, \cdots, S_n$ where each $S_i \subseteq \Sigma$ has at most l elements, the number of codewords $c \in C$ for which $c_i \in S_i$ for at least ζn positions $i \in \{1, 2, \dots, n\}$ is at most L.

A code $C \subseteq \Sigma^n$ is said to (ζ, l)-list recoverable in polynomial time if it is $(\zeta, l, L(n))$-list recoverable for some polynomially bounded function $L(\cdot)$, and moreover there is a polynomial time algorithm to find the at most $L(n)$ codewords that are solutions to any $(\zeta, l, L(n))$-list recovering instance.

Note that when $l = 1$, $(\zeta, 1, \cdot)$-list recovering is the same as list decoding up to a $(1 - \zeta)$ fraction of errors. Guruswami and Rudra have the following result for list recovering.

Theorem 2 ([5]) *For every integer $l \geq 1$, for all R, $0 < R < 1$ and $\epsilon > 0$, and for every prime p, there is an explicit family of folded Reed–Solomon codes over fields of characteristic p that have rate at least R and which can be $(R + \epsilon, l)$-list recovered in polynomial time. The alphabet size of a code of block length N in the family is $(N/\epsilon^2)^{O(\epsilon^{-2}\log l/(1-R))}$.*

Applications

To get within ϵ of capacity, the codes in Theorem 1 have alphabet size $N^{\Omega(1/\epsilon^2)}$ where N is the block length. By concatenating folded RS codes of rate close to 1 (that are list recoverable) with suitable inner codes followed by redistribution of symbols using an expander graph (similar to a construction for linear-time unique decodable codes in [3]), one can get within ϵ of capacity with codes over an alphabet of size $2^{O(\epsilon^{-4}\log(1/\epsilon))}$. A counting argument shows that codes that can be list decoded efficiently to within ϵ of the capacity need to have an alphabet size of $2^{\Omega(1/\epsilon)}$.

For binary codes, the list decoding capacity is known to be $\rho_{\text{bin}}(R) = H^{-1}(1 - R)$ where $H(\cdot)$ denotes the binary entropy function. No explicit constructions of binary codes that approach this capacity are known. However, using the Folded RS codes of Guruswami Rudra in a natural concatenation scheme, one can obtain polynomial time constructable binary codes of rate R that can be list decoded up to a fraction $\rho_{\text{Zyab}}(R)$ of errors, where $\rho_{\text{Zyab}}(R)$ is the "Zyablov bound".

See [5] for more details.

Open Problems

The work of Guruswami and Rudra could be improved with respect to some parameters. The size of the list needed to perform list decoding to a radius that is within ϵ of capacity grows as $N^{O(\epsilon^{-1}\log(1/R))}$ where N and R are the block length and the rate of the code respectively. It remains an open question to bring this list size down to a constant independent of n (the existential random coding arguments work with a list size of $O(1/\epsilon)$). The alphabet size needed to approach capacity was shown to be a constant independent of N. However, this involved a brute-force search for a rather large (inner) code, which translates to a construction time of about $N^{O(\epsilon^{-2}\log(1/\epsilon))}$ (instead of the ideal construction time where the exponent of N does not depend on ϵ). Obtaining a "direct" algebraic construction over a constant-sized alphabet, such as the generalization of the Parvaresh-Vardy framework to algebraic-geometric codes in [4], might help in addressing these two issues.

Finally, constructing binary codes that approach list decoding capacity remains open.

Cross References

▶ Decoding Reed–Solomon Codes
▶ Learning Heavy Fourier Coefficients of Boolean Functions
▶ LP Decoding

Recommended Reading

1. Elias, P.: List decoding for noisy channels. Technical Report 335, Research Laboratory of Electronics MIT (1957)
2. Elias, P.: Error-correcting codes for list decoding. IEEE Trans. Inf. Theory **37**, 5–12 (1991)
3. Guruswami, V., Indyk, P.: Linear-time encodable/decodable codes with near-optimal rate. IEEE Trans. Inf. Theory **51**(10), 3393–3400 (2005)
4. Guruswami, V., Patthak, A.: Correlated Algebraic-Geometric codes: Improved list decoding over bounded alphabets. In: Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp. 227–236, Berkley, October 2006
5. Guruswami, V., Rudra, A.: Explicit capacity-achieving list-decodable codes. In: Proceedings of the 38th Annual ACM Symposium on Theory of Computing, pp. 1–10. Seattle, May 2006
6. Guruswami, V., Sudan, M.: Improved decoding of Reed–Solomon and algebraic-geometric codes. IEEE Trans. Inf. Theory **45**, 1757–1767 (1999)
7. Parvaresh, F., Vardy, A.: Correcting errors beyond the Guruswami—Sudan radius in polynomial time. In: Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science, pp. 285–294. Pittsburgh, 2005
8. Sudan, M.: Decoding of Reed—Solomon codes beyond the error-correction bound. J Complex. **13**(1), 180–193 (1997)
9. Wozencraft, J.M.: List Decoding. Quarterly Progress Report, Research Laboratory of Electronics. MIT **48**, 90–95 (1958)
10. Zyablov, V.V., Pinsker, M.S.: List cascade decoding. Probl. Inf. Trans. **17**(4), 29–34 (1981) (in Russian); pp. 236–240 (in English) (1982)

List Scheduling

1966; Graham

LEAH EPSTEIN
Department of Mathematics, University of Haifa, Haifa, Israel

Keywords and Synonyms

Online scheduling on identical machines

Problem Definition

The paper of Graham [8] was published in the 1960s. Over the years it served as a common example of online algorithms (though the original algorithm was designed as a simple approximation heuristic). The following basic setting is considered.

A sequence of n jobs is to be assigned to m identical machines. Each job should be assigned to one of the machines. Each job has a size associated with it, which can be seen as its processing time or its load. The load of a machine is the sum of sizes of jobs assigned to it. The goal is to minimize the maximum load of any machine, also called the makespan. We refer to this problem as JOB SCHEDULING.

If jobs are presented one by one, and each job needs to be assigned to a machine in turn, without any knowledge of future jobs, the problem is called online. Online algorithms are typically evaluated using the (absolute) *competitive ratio*, which is similar to the *approximation ratio* of approximation algorithms. For an algorithm $\mathcal{A}$, we denote its cost by $\mathcal{A}$ as well. The cost of an optimal offline algorithm that knows the complete sequence of jobs is denoted by OPT. The competitive ratio of an algorithm $\mathcal{A}$ is the infimum $\mathcal{R} \geq 1$ such that for any input, $\mathcal{A} \leq \mathcal{R} \cdot \mathrm{OPT}$.

Key Results

In paper [8], Graham defines an algorithm called LIST SCHEDULING (LS). The algorithm receives jobs one by one. Each job is assigned in turn to a machine which has a minimal current load. Ties are broken arbitrarily.

The main result is the following.

Theorem 1 LS *has a competitive ratio of* $2 - \frac{1}{m}$.

Proof. Consider a schedule created for a given sequence. Let ℓ denote a job that determines the makespan (that is, the last job assigned to a machine i that has a maximum load), let L denote its size, and let X denote the total size of all other jobs assigned to i. At the time when L was assigned to i, this was a machine of minimum load. Therefore, the load of each machine is at least X. The makespan of an optimal schedule (i. e., a schedule that minimizes the makespan) is the cost of an optimal offline algorithm and thus is denoted by OPT. Let P be the sum of all job sizes in the sequence.

The two following simple lower bounds on OPT can be obtained.

$$\mathrm{OPT} \geq L\,. \tag{1}$$

$$\mathrm{OPT} \geq \frac{P}{m} \geq \frac{m \cdot X + L}{m} = X + \frac{L}{m}\,. \tag{2}$$

Inequality (1) follows from the fact that {OPT} needs to run job ℓ and thus at least one machine has a load of at least L. The first inequality in (2) is due to the fact that at least one machine receives at least a fraction of $\frac{1}{m}$ of the total size of jobs. The second inequality in (2) follows from the comments above on the load of each machine.

This proves that the makespan of the algorithm, $X + L$ can be bounded as follows.

$$1X + L \leq \text{OPT} + \frac{m-1}{m}L \leq \text{OPT} + \frac{m-1}{m}\text{OPT}$$
$$= (2 - 1/m)\,\text{OPT}\,. \quad (3)$$

The first inequality in (3) follows from (2) and the second one from (1).

To show that the analysis is tight, consider $m(m-1)$ jobs of size 1 followed by a single job of size m. After the smaller jobs arrive, LS obtains a balanced schedule in which every machine has a load of $m - 1$. The additional job increases the makespan to $2m - 1$. However, an optimal offline solution would be to assign the smaller jobs to $m - 1$ machines, and the remaining job to the remaining machine, getting a load of m.

A natural question was whether this bound is best possible. In a later paper, Graham [9] showed that applying LS with a sorted sequence of jobs (by non-increasing order of sizes) actually gives a better upper bound of $\frac{4}{3} - \frac{1}{3m}$ on the approximation ratio. A polynomial time approximation scheme was given by Hochbaum and Shmoys in [10]. This is the best offline result one could hope for as the problem is known to be NP-hard in the strong sense.

As for the online problem, it was shown in [5] that no (deterministic) algorithm has a smaller competitive ratio than $2 - \frac{1}{m}$, for the cases $m = 2$ and $m = 3$. On the other hand, it was shown in a sequence of papers that an algorithm with a smaller competitive ratio can be found for any $m \geq 4$, and even algorithms with a competitive ratio that does not approach 2 for large m were designed.

The best such result is by Fleischer and Wahl [6], who designed a 1.9201-competitive algorithm. Lower bounds of 1.852 and 1.85358 on the competitive ratio of any online algorithm were shown in [1,7]. Rudin [13] claimed a better lower bound of 1.88.

Applications

As the study of approximation algorithms and specifically online algorithms continued, the analysis of many scheduling algorithms used similar methods to the proof above. Below, several variants of the problem where almost the same proof as above gives the exact same bound are mentioned.

Load Balancing of Temporary Tasks

In this problem the sizes of jobs are seen as loads. Time is a separate axis. The input is a sequence of events, where every event is an arrival or a departure of a job. The set of active jobs at time t is the set of jobs that have already arrived at this time and have not departed yet. The cost of an algorithm at a time t is its makespan at this time. The cost of an algorithm is its maximum cost over time. It turns out that the analysis above can be easily adapted for this model as well. It is interesting to note that in this case the bound $2 - \frac{1}{m}$ is actually best possible, as shown in [2].

Scheduling with Release Times and Precedence Constraints

In this problem, the sizes represent processing times of jobs. Various versions have been studied. Jobs may have designated release times, which are the times when these jobs become available for execution. In the online scenario, each job arrives and becomes known to the algorithm only at its release time. Some precedence constraints may also be specified, defined by a partial order on the set of jobs. Thus, a job can be run only after its predecessors complete their execution. In the online variant, a job becomes known to the algorithm only after its predecessors have been completed. In these cases, LS acts as follows. Once a machine becomes available, a waiting job that arrived earliest is assigned to it. (If there is no waiting job, the machine is idle until a new job arrives.)

The upper bound of $2 - \frac{1}{m}$ on the competitive ratio can be proved using a relation between the cost of an optimal schedule, and the amount of time when at least one machine is idle. (See [14] for details).

This bound is tight for several cases. For the case where there are release times, no precedence constraints, and processing times (sizes) are not known upon arrival, Shmoys, Wein, and Williamson [15] proved a lower bound of $2 - \frac{1}{m}$. For the case where there are only precedence constraints (no release times, and sizes of jobs are known upon arrival), a lower bound of the same value appeared in [4]. Note that the case with clairvoyant scheduling (i. e., sizes of jobs are known upon arrival), release times, and no precedence constraints is not settled. For $m = 2$ it was shown by Noga and Seiden [11] that the tight bound is $(5 - \sqrt{5})/2 \approx 1.38198$, and the upper bound is achieved using an algorithm that applies waiting with idle machines rather than scheduling a job as soon as possible, as done by LS.

Open Problems

The most challenging open problem is to find the best possible competitive ratio for this basic online problem of job scheduling. The gap between the upper bound and the lower bound is not large, yet it seems very difficult to find the exact bound. A possibly easier question would be to find the best possible competitive ratio for $m = 4$. A lower bound of $\sqrt{3} \approx 1.732$ has been shown by [12] and the currently known upper bound is 1.733 by [3]. Thus, it may be the case that this bound would turn out to be $\sqrt{3}$.

Recommended Reading

1. Albers, S.: Better bounds for online scheduling. SIAM J. Comput. **29**(2), 459–473 (1999)
2. Azar, Y., Epstein, L.: On-line load balancing of temporary tasks on identical machines. SIAM J. Discret. Math. **18**(2), 347–352 (2004)
3. Chen, B., van Vliet, A., Woeginger, G.J.: New lower and upper bounds for on-line scheduling. Oper. Res. Lett. **16**, 221–230 (1994)
4. Epstein, L.: A note on on-line scheduling with precedence constraints on identical machines. Inf. Process. Lett. **76**, 149–153 (2000)
5. Faigle, U., Kern, W., Turán, G.: On the performane of online algorithms for partition problems. Acta Cybern. **9**, 107–119 (1989)
6. Fleischer, R., Wahl, M.: On-line scheduling revisited. J. Sched. **3**, 343–353 (2000)
7. Gormley, T., Reingold, N., Torng, E., Westbrook, J.: Generating adversaries for request-answer games. In: Proc. of the 11th Symposium on Discrete Algorithms. (SODA2000), pp. 564–565 (2000)
8. Graham, R.L.: Bounds for certain multiprocessing anomalies. Bell Syst. Techn. J. **45**, 1563–1581 (1966)
9. Graham, R.L.: Bounds on multiprocessing timing anomalies. SIAM J. Appl. Math. **17**, 263–269 (1969)
10. Hochbaum, D.S., Shmoys, D.B.: Using dual approximation algorithms for scheduling problems: theoretical and practical results. J. ACM **34**(1), 144–162 (1987)
11. Noga, J., Seiden, S.S.: An optimal online algorithm for scheduling two machines with release times. Theor. Comput. Sci. **268**(1), 133–143 (2001)
12. Rudin III, J.F., Chandrasekaran, R.: Improved bounds for the online scheduling problem. SIAM J. Comput. **32**, 717–735 (2003)
13. Rudin III, J.F.: Improved bounds for the online scheduling problem. Ph. D. thesis, The University of Texas at Dallas (2001)
14. Sgall, J.: On-line scheduling. In: Fiat, A., Woeginger, G.J. (eds.) Online Algorithms: The State of the Art, pp. 196–231. Springer (1998)
15. Shmoys, D.B., Wein, J., Williamson, D.P.: Scheduling parallel machines on-line. SIAM J. Comput. **24**, 1313–1331 (1995)

Load Balancing

1994; Azar, Broder, Karlin
1997; Azar, Kalyanasundaram, Plotkin, Pruhs, Waarts

LEAH EPSTEIN
Department of Mathematics, University of Haifa, Haifa, Israel

Keywords and Synonyms

Online scheduling of temporary tasks

Problem Definition

Load balancing of temporary tasks is a online problem. In this problem, arriving tasks (or jobs) are to be assigned to processors, which are also called machines. In this entry, deterministic online load balancing of temporary tasks with unknown duration is discussed. The input sequence consists of departures and arrivals of tasks. If the sequence consists of arrivals only, the tasks are called permanent. Events happen one by one, so that the next event appears after the algorithm completed dealing with the previous event.

Clearly, the problem with temporary tasks is different from the problem with permanent tasks. One such difference is that for permanent tasks, the maximum load is always achieved in the end of the sequence. For temporary tasks this is not always the case. Moreover, the maximum load may be achieved at different times for different algorithms.

In the most general model, there are m machines $1, \ldots, m$. The information of an arriving job j is a vector p_j of length m, where p_j^i is the load or size of job j, if it is assigned to machine i. As stated above, each job is to be assigned to a machine before the next arrival or departure. The load of a machine i at time t is denoted by L_i^t and is the sum of the loads (on machine i) of jobs which are assigned to machine i, that arrived by time t and did not depart by this time. The goal is to minimize the maximum load of any machine over all times t. This machine model is known as *unrelated machines* (see [3] for a study of the load balancing problem of permanent tasks on unrelated machines). Many more specific models were defined. In the sequel a few such models are described.

For an algorithm $\mathcal{A}$, denote its cost by $\mathcal{A}$ as well. The cost of an optimal offline algorithm that knows the complete sequence of events in advance is denoted by OPT. Load balancing is studied in terms of the (absolute) competitive ratio. The competitive ratio of $\mathcal{A}$ is the infimum $\mathcal{R}$ such that for any input, $\mathcal{A} \leq \mathcal{R} \cdot \text{OPT}$. If the competitive ratio of an online algorithm is at most C it is also called C-competitive.

Uniformly related machines [3,12] are machines with speeds associated with them, thus machine i has speed s_i and the information that a job j needs to provide upon its arrival is just its size, or the load that it incurs on a unit speed machine, which is denoted by p_j. Then let $p_j^i = p_j/s_i$. If all speeds are equal, this results in *identical machines* [15].

Restricted assignment [8] is a model where each job may be run only on a subset of the machines. A job j is associated with running time which is the time to run it on any of its permitted machines M_j. Thus if $i \in M_j$ then $p^i_j = p_j$ and otherwise $p^i_j = \infty$.

Key Results

The known results in all four models are surveyed below.

Identical Machines

Interestingly, the well known algorithm of Graham [15], LIST SCHEDULING, which is defined for identical machines, is valid for temporary tasks as well as permanent tasks. This algorithm greedily assigns a new job to the least loaded machine. The competitive ratio of this algorithm is $2 - 1/m$, which is best possible (see [5]). Note that the competitive ratio is the same as for permanent tasks, but for permanent tasks, it is possible to achieve a competitive ratio which does not tend to 2 for large m, see e. g. [11].

Uniformly Related Machines

The situation for uniformly related machines is not very different. In this case, the algorithms of Aspnes et al. [3] and of Berman et al. [12] cannot applied as they are, and some modifications are required. The algorithm of Azar et al. [7] has competitive ratio of at most 20 and it is based on the general method introduced in [3]. The algorithm of [3] keeps a guess value λ, which is an estimation of the cost of an optimal offline algorithm OPT. An invariant that must be kept is $\lambda \leq 2\text{OPT}$. At each step, a procedure is applied for some value of λ (which can be initialized as the load of the first job on the fastest machine). The procedure for a given value of λ is applied until it fails, and some job cannot be assigned while satisfying all conditions. The procedure is designed so that if it fails, then it must be the case that $\text{OPT} > \lambda$, the value of λ is doubled, and the procedure is re-invoked for the new value, ignoring all assignments that were done for small values of λ. This method is called doubling, and results in an algorithm with a competitive ratio which is at most four times the competitive ratio achieved by the procedure. The procedure for a given λ acts as follows. Let c be a target competitive ratio for the procedure. The machines are sorted according to speed. Each job is assigned to the first machine in the sorted order such that the job is assignable to it. A job j arriving at time t is assignable to machine i if $p_j/s_i \leq \lambda$ and $L^{t-1}_i + p_j/s_i \leq c\lambda$. It is shown in [7] that $c = 5$ allows the algorithm to succeed in the assignment of all jobs (i. e., to have at least one assignable machine for each job), as long as $\text{OPT} \leq \lambda$. Note that the constant c for permanent tasks used in [3] is 2. As for lower bounds, it is shown in [7] that the competitive ratio $\mathcal{R}$ of any algorithm satisfies $\mathcal{R} \geq 3 - o(1)$. The upper bound has been improved to $6 + 2\sqrt{5} \approx 10.47$ by Bar-Noy et al. [9].

Restricted Assignment

As for restricted assignment, temporary tasks make this model much more difficult than permanent tasks. The competitive ratio $O(\log m)$ which is achieved by a simple greedy algorithm (see [8]) does not hold in this case. In fact, the competitive ratio of this algorithm becomes $\Omega(m^{\frac{2}{3}})$ [4]. Moreover, in the same paper, a lower bound of $\Omega\sqrt{m}$ on the competitive ratio of any algorithm was shown. The construction was quite involved, however, Ma and Plotkin [16] gave a simplified construction which yields the same result.

The construction of [16] selects a value p which is the largest integer that satisfies $p + p^2 \leq m$. Clearly $p = \Theta(\sqrt{m})$. The lower bound uses two sets of machines, p machines which are called "the small group", and p^2 machines which are called "the large group". The construction consists of p^2 phases, each of which consists of p jobs and is dedicated to one machine in the large group. In phase i, job k of this phase can run either on the k-th machine of the small group, or the i-th machine of the large group. After this arrival, only one of these p jobs does not depart. An optimal offline algorithm assigns all jobs in each phase to the small group, except for the one job that will not depart. Thus when the construction is completed, it has one job on each machine of the large group. The maximum load ever achieved by OPT is 1. However, the algorithm does not know at each phase which job will not depart. If no job is assigned to the small group in phase i, then the load of machine i becomes p. Otherwise, a job that the algorithm assigns to the small group is chosen as the one that will not depart. In this way, after p phases, a total load of p^2 is accumulated on the small group, which means that at least one machine there has load p. This completed the construction.

An alternative algorithm called ROBIN HOOD was designed in [7]. This algorithm keeps a lower bound on OPT, which is the maximum between the following two functions. The first one is the maximum average machine load over time. The second is the maximum job size that has ever arrived. Denote this lower bound at time t (after t events have happened) by B^t. A machine i is called rich at time t if $L^t_i \geq \sqrt{m}B^t$. Otherwise it is called poor. The windfall time of a rich machine i at time t is the time t' such that i is poor at time $t' - 1$ and rich at times $t', \ldots, t$, i. e., the last time that machine i became rich. Clearly, machines can become poor due to an update of B^t or departure of jobs. A machine can become rich due to arrival of jobs that are assigned to it.

The algorithm assigns a job j to a poor machine in $M(j)$, if such a machine exists. Otherwise, j is assigned to the machine in $M(j)$ with the most recent windfall time. The analysis makes use of the fact that at most $\sqrt{m}$ machines can be rich simultaneously.

Note that for small values of m ($m \le 5$), the competitive ratio of the greedy algorithm is still best possible, as shown in [1]. In this paper it was shown that these bounds are $(m+3)/2$ for $m = 3, 4, 5$. It is not difficult to see that for $m = 2$, the best bound is 2.

Unrelated Machines

The most extreme difference occurs for unrelated machines. Unlike the case of permanent tasks, where an upper bound of $O(\log m)$ can be achieved [3], it was shown in [2] that any algorithm has a competitive ratio of $\Omega(m/\log m)$. Note that a trivial algorithm, which assigns each job to the machine where it has a minimum load, has competitive ratio of at most m [3].

Applications

A similar model is known for the bin packing problem as well. In this problem, the sequence consists of arrivals and departures items of sizes in $(0, 1]$. The goal is to keep a partition of the currently existing items into subsets (bins), where the sum of items in each subset is at most 1. The cost is the maximum number of bins ever used simultaneously. This problem was studied in [13,14].

In [10], an hierarchical model was studied. This is a special case of restricted assignment where for each job j, $M(j)$ is a prefix of the machines. They showed that even for temporary tasks an algorithm of constant competitive ratio exists for this model.

In [6], which studied resource augmentation in load balancing, temporary tasks were considered as well. Resource augmentation is a type of analysis where the online algorithm is compared to an optimal offline algorithm which has less machines.

Open Problems

Small gaps still remain for both uniformly related machines, and for unrelated machines. For unrelated machines is could be interesting to find if there exists an algorithm of competitive ratio $o(m)$, or whether the simple algorithm stated above has optimal competitive ratio (up to a multiplicative factor).

Cross References

See ► List Scheduling for more information on identical machines and [15].

Recommended Reading

1. Armon, A., Azar, Y., Epstein, L., Regev, O.: On-line restricted assignment of temporary tasks with unknown durations. Inf. Process. Lett. **85**(2), 67–72 (2003)
2. Armon, A., Azar, Y., Epstein, L., Regev, O.: Temporary tasks assignment resolved. Algorithmica **36**(3), 295–314 (2003)
3. Aspnes, J., Azar, Y., Fiat, A., Plotkin, S., Waarts, O.: On-line load balancing with applications to machine scheduling and virtual circuit routing. J. ACM **44**, 486–504 (1997)
4. Azar, Y., Broder, A.Z., Karlin, A.R.: On-line load balancing. Theor. Comput. Sci. **130**, 73–84 (1994)
5. Azar, Y., Epstein, L.: On-line load balancing of temporary tasks on identical machines. SIAM J. Discret. Math. **18**(2), 347–352 (2004)
6. Azar, Y., Epstein, L., van Stee, R.: Resource augmentation in load balancing. J. Sched. **3**(5), 249–258 (2000)
7. Azar, Y., Kalyanasundaram, B., Plotkin, S., Pruhs, K., Waarts, O.: On-line load balancing of temporary tasks. J. Algorithms **22**(1), 93–110 (1997)
8. Azar, Y., Naor, J., Rom, R.: The competitiveness of on-line assignments. J. Algorithms **18**, 221–237 (1995)
9. Bar-Noy, A., Freund, A., Naor, J.: New algorithms for related machines with temporary jobs. J. Sched. **3**(5), 259–272 (2000)
10. Bar-Noy, A., Freund, A., Naor, J.: On-line load balancing in a hierarchical server topology. SIAM J. Comput. **31**, 527–549 (2001)
11. Bartal, Y., Fiat, A., Karloff, H., Vohra, R.: New algorithms for an ancient scheduling problem. J. Comput. Syst. Sci. **51**(3), 359–366 (1995)
12. Berman, P., Charikar, M., Karpinski, M.: On-line load balancing for related machines. J. Algorithms **35**, 108–121 (2000)
13. Chan, W.-T., Wong, P.W.H., Yung, F.C.C.: On dynamic bin packing: an improved lower bound and resource augmentation analysis. In: Proc. of the 12th Annual International Conference on Computing and Combinatorics (COCOON2006), 2006, pp. 309–319
14. Coffman, E.G., Garey, M.R., Johnson, D.S.: Dynamic bin packing. SIAM J. Comput. **12**(2), 227–258 (1983)
15. Graham, R.L.: Bounds for certain multiprocessing anomalies. Bell Syst. Tech. J. **45**, 1563–1581 (1966)
16. Ma, Y., Plotkin, S.: Improved lower bounds for load balancing of tasks with unknown duration. Inf. Process. Lett. **62**, 31–34 (1997)

Local Alignment (with Affine Gap Weights)

1986; Altschul, Erickson

Stephen F. Altschul[1,2], Bruce W. Erickson[1], Henry Leung[2]

[1] The Rockefeller University, New York, NY, USA

[2] Department of Applied Mathematics, MIT, Cambridge, MA, USA

Keywords and Synonyms

Pairwise local alignment with affine gap weight

Problem Definition

The pairwise local alignment problem is concerned with identification of a pair of similar substrings from two molecular sequences. This problem has been studied in computer science for four decades. However, most problem models were generally not biologically satisfying or interpretable before 1974. In 1974, Sellers developed a metric measure of the similarity between molecular sequences. Waterman et al. (1976) generalized this metric to include deletions and insertions of arbitrary length which represent the minimum number of mutational events required to convert one sequence into another.

Given two sequences S and T, a pairwise alignment is a way of inserting space characters '_' in S and T to form sequences S' and T' respectively with the same length. There can be different alignments of two sequences. The score of an alignment is measured by a scoring metric $\delta(x, y)$. At each position i where both x and y are not spaces, the similarity between $S'[i]$ and $T'[i]$ is measured by $\delta(S'[i], T'[j])$. Usually, $\delta(x, y)$ is positive when x and y are the same and negative when x and y are different. For positions with consecutive space characters, the alignment scores of the space characters are not considered independently; this is because inserting or deleting a long region in molecular sequences is more likely to occur than inserting or deleting several short regions. Smith and Waterman use an affine gap penalty to model the similarity at positions with space characters. They define a consecutive substring with spaces in S' or T' be a gap. For each length l gap, they give a linear penalty $W_k = W_s + l \times W_p$ for some predefined positive constants W_s and W_p. The score of an alignment is the sum of the score at each position i minus the penalties of each gap. For example, the alignment score of the following alignment is $\delta(G, G) + \delta(C, C) + \delta(C, C) + \delta(U, C) + \delta(G, G) - (W_s + 2 \times W_p)$.

$$S: GCCAUUG$$
$$T: GCC\_\_CG$$

The optimal global alignment of sequences S and T is the alignment of S and T with the maximum alignment score.

Sometimes we want to know whether sequences S and T contain similar substrings instead of whether S and T are similar. In this case, they solve the pairwise local alignment problem, which wants to find a substring U in S and another substring V in T such that the global alignment score of U and V is maximized.

Pairwise Local Alignment Problem

Input: Two sequences $S[1..n]$ and $T[1..m]$.
Output: A substring U in S and a substring V in T such that the optimal global alignment of U and V is maximized.

Key Results

The pairwise local alignment problem can be solved in $O(mn)$ time and $O(mn)$ space by dynamic programming. The algorithm needs to fill in the 4 $m \times n$ tables H, H_N, H_S and H_T, where each entry takes constant time. The individual meanings of these 4 tables are as follows.

$\boldsymbol{H(i,j)}$:	maximum score of the global alignment of U and V over all suffixes U in $S[1..i]$ and all suffixes V in $T[1..j]$.
$\boldsymbol{H_N(i,j)}$:	maximum score of the global alignment of U and V over all suffixes U in $S[1..i]$ and all suffixes V in $T[1..j]$, with the restriction that $S[i]$ and $T[j]$ must be aligned.
$\boldsymbol{H_S(i,j)}$:	maximum score of the global alignment of U and V over all suffixes U in $S[1..i]$ and all suffixes V in $T[1..j]$, with $S[j]$ aligned with a space character.
$\boldsymbol{H_T(i,j)}$:	maximum score of the global alignment of U and V over all suffixes U in $S[1..i]$ and all suffixes V in $T[1..j]$, with $T[j]$ aligned with a space character.

The optimal local alignment score of S and T will be $\max\{H(i, j)\}$ and the local alignment of S and T can be found by tracking back table H.

In the tables, each entry can be filled in by the following recursion in constant time.

Basic Step:

$$\begin{aligned} H(i,0) = H(0,j) &= 0\,, & 0 \leqslant i \leqslant n\,,\ 0 \leqslant i \leqslant m \\ H_N(i,0) = H_N(0,j) &= -\infty\,, & 0 \leqslant i \leqslant n\,,\ 0 \leqslant i \leqslant m \\ H_s(i,0) = H_T(0,j) &= W_s + W_p\,, & 0 \leqslant i \leqslant n\,,\ 0 \leqslant i \leqslant m \\ H_s(0,j) = H_T(i,0) &= -\infty\,, & 0 \leqslant i \leqslant n\,,\ 0 \leqslant i \leqslant m \end{aligned}$$

Recursion Step:

$$\begin{aligned} H(i,j) &= \max\{H_N(i,j), H_s(i,j), H_T(i,j), 0\}\,, \\ &\qquad 1 \leqslant i \leqslant n\,,\quad 1 \leqslant i \leqslant m \\ H_N(i,j) &= H(i-1,j-1) + \delta(S[i], T[j])\,, \\ &\qquad 1 \leqslant i \leqslant n\,,\quad 1 \leqslant i \leqslant m \end{aligned}$$

$$H_s(i,j) = \max\{H(i-1,j) - (W_s + W_p),$$
$$H_S(i-1,j) - W_p\}\,,$$
$$1 \leqslant i \leqslant n\,,\quad 1 \leqslant i \leqslant m$$
$$H_T(i,j) = \max\{H(i,j-1) - (W_s + W_p),$$
$$H_T(i,j-1) - W_p\}\,,$$
$$1 \leqslant i \leqslant n\,,\quad 1 \leqslant i \leqslant m$$

Applications

Local alignment with affine gap penalty can be used for protein classification, phylogenetic footprinting and identification of functional sequence elements.

URL to Code

http://bioweb.pasteur.fr/seqanal/interfaces/water.html

Cross References

► Efficient Methods for Multiple Sequence Alignment with Guaranteed Error Bounds
► Local Alignment (with Concave Gap Weights)

Recommended Reading

1. Allgower, E.L., Schmidt, P.H.: An Algorithm for Piecewise-Linear Approximation of an Implicitly Defined Manifold. SIAM J. Num. Anal. **22**, 322–346 (1985)
2. Altschul, S.F., Gish, W., Miller, W., Myers, E.W., Lipman, D.J.: Basic Local Alignment Search Tool. J. Mol. Biol. **215**, 403–410 (1990)
3. Chao, K.M., Miller, W.: Linear-space algorithms that build local alignments from fragments. Algorithmica **13**, 106–134 (1995)
4. Myers, E.W., Miller, W.: Optimal Alignments in Linear Space. Bioinformatics **4**, 11–17 (1988)
5. Gusfield, D.: Algorithms on Strings, Trees and Sequences. Cambridge University Press, Cambridge (1999). ISBN 052158519
6. Ma, B., Tromp, J., Li, M.: PatternHunter: Faster and More Sensitive Homology Search. Bioinformatics **18**, 440–445 (2002)
7. Needleman, S.B., Wunsch, C.D.: A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins. J. Mol. Biol. **48**, 443–453 (1970)
8. Pearson, W.R., Lipman, D.J.: Improved Tools for Biological Sequence Comparison. Proc. Natl. Acad. Sci. USA **85**, 2444–2448 (1988)
9. Sellers, P.H.: On the Theory and Computation of Evolutionary Distances. SIAM J. Appl. Math. **26**, 787–793 (1974)
10. Smith, T.F., Waterman, M.S.: Identification of Common Molecular Subsequences. J. Mol. Biol. **147**, 195–197 (1981)

Local Alignment (with Concave Gap Weights)

1988; Miller, Myers

S. M. YIU
Department of Computer Science, The University of Hong Kong, Hong Kong, China

Keywords and Synonyms

Sequence alignment; Pairwise local alignment

Problem Definition

This work of Miller and Myers [9] deals with the problem of pairwise sequence alignment in which the distance measure is based on the gap penalty model. They proposed an efficient algorithm to solve the problem when the gap penalty is a concave function of the gap length.

Let X and Y be two strings (sequences) of alphabet Σ. The pairwise alignment $\mathcal{A}$ of X and Y maps X, Y into strings X', Y' that may contain spaces (not in Σ) such that (1) $|X'| = |Y'| = \ell$; (2) removing spaces from X' and Y' returns X and Y, respectively; and (3) for any $1 \le i \le \ell$, $X'[i]$ and $Y'[i]$ cannot be both spaces where $X'[i]$ denotes the ith character in X'.

To evaluate the quality of an alignment, there are many different measures proposed (e. g. edit distance, scoring matrix [11]). In this work, they consider the *gap penalty* model.

A *gap* in an alignment $\mathcal{A}$ of X and Y is a maximal substring of contiguous spaces in either X' or Y'. There are gaps and aligned characters (both $X'[i]$ and $Y'[i]$ are not space) in an alignment. The score for a pair of aligned characters is based on a distance function $\delta(a,b)$ where $a, b \in \Sigma$. Usually δ is a metric, but this assumption is not required in this work. The penalty of a gap of length k is based on a non-negative function $W(k)$. The score of an alignment is the sum of the scores of all aligned characters and gaps. An alignment is *optimal* if its score is the minimum possible.

The penalty function $W(k)$ is *concave* if $\triangle W(k) \ge \triangle W(k+1)$ for all $k \ge 1$, where $\triangle W(k) = W(k+1) - W(k)$.

The penalty function $W(k)$ is *affine* if $W(k) = a + bk$ where a, b are constants. Affine function is a special case of concave function. The problem for affine gap penalty has been considered in [1,6].

The penalty function $W(k)$ is a *P-piece affine curve* if the domain of W can be partitioned into P intervals, $(\tau_1 = 1, \chi_1), (\tau_2, \chi_2), \ldots, (\tau_p, \chi_p = \infty)$, where $\tau_i = \chi_{i-1} + 1$ for all $1 < i \le p$, such that for each interval, the values of W follow an affine function. More precisely, for any $k \in (\tau_i, \chi_i)$, $W(k) = a_i + b_i k$ for some constants a_i, b_i.

Problem

INPUT: Two strings X and Y, the scoring function δ, and the gap penalty function $W(k)$.
OUTPUT: An optimal alignment of X and Y.

Key Results

Theorem 1 *If $W(k)$ is concave, they provide an algorithm for computing an optimal alignment that runs in $O(n^2 \log n)$ time where n is the length of each string and uses $O(n)$ expected space.*

Corollary 1 *If $W(k)$ is an affine function, the same algorithm runs in $O(n^2)$ time.*

Theorem 2 *For some special types of gap penalty functions, the algorithm can be modified to run faster.*

- *If $W(k)$ is a P-piece affine curve, the algorithm can be modified to run in $O(n^2 \log P)$ time.*
- *For logarithmic gap penalty function, $W(k) = a + b \log k$, the algorithm can be modified to run in $O(n^2)$ time.*
- *If $W(k)$ is a concave function when $k > K$, the algorithm can be modified to run in $O(K + n^2 \log n)$ time.*

Applications

Pairwise sequence alignment is a fundamental problem in computational biology. Sequence similarity usually implies functional and structural similarity. So, pairwise alignment can be used to check whether two given sequences have similar functions or structures and to predict functions of newly identified DNA sequence. One can refer to Gusfield's book for some examples on the importance of sequence alignment (pp. 212–214 of [7]).

The alignment problem can be further divided into the *global* alignment problem and the *local* alignment problem. The problem defined here is the global alignment problem in which the whole input strings are required to align with each other. On the other hand, for local alignment, the main interest lies in identifying a substring from each of the input strings such that the alignment score of the two substrings is the minimum among all possible substrings. Local alignment is useful in aligning sequences that are not similar, but contain a region that are highly conserved (similar). Usually this region is a functional part (domain) of the sequences. Local alignment is particularly useful in comparing proteins. Proteins in the same family from different species usually have some functional domains that are highly conserved while the other parts are not similar at all. Examples are the homeobox genes [10] for which the protein sequences are quite different in each species except the functional domain *homeodomain*.

Conceptually, the alignment score is used to capture the evolutionary distance between the two given sequences. Since a gap of more than one space can be created by a single mutational event, so considering a gap of length k as a unit instead of k different point mutation may be more appropriate in some cases. However, which gap penalty function should be used is a difficult question to answer and sometimes depend on the actual applications. Most applications, such as BLAST, uses the affine gap penalty which is still the dominate model in practice. On the other hand, Benner et al. [2] and Gu and Li [13] suggested to use the logarithmic gap penalty in some cases. Whether using a concave gap penalty function in general is meaningful is still an open issue.

Open Problem

Note that the results of this paper have been independently obtained by Galil and Giancarlo [5] and for affine gap penalty, Gotoh [6] also gave an $O(n^2)$ algorithm for solving the alignment problem. In [4], Eppstein gave a faster algorithm that runs in $O(n^2)$ time for solving the same sequence alignment problem with concave gap penalty function. Whether a subquadratic algorithm exists for solving this problem remains open. As a remark, subquadratic algorithms do exist for solving the sequence alignment problem if the measure is not based on the gap penalty model, but is computed as $\sum_{i=1}^{\ell} \delta(X1'[i], Y'[i])$ based only on a scoring function $\delta(a, b)$ where $a, b \in \Sigma \cup \{\_\}$ where '_' represents the space [3,8].

Experimental Results

They have performed some experiments to compare their algorithm with Waterman's $O(n^3)$ algorithm [12] on a number of different concave gap penalty functions. Artificial sequences are generated for the experiments. Results from their experiments lead to their conjectures that Waterman's method runs in $O(n^3)$ time when the two given strings are very similar or the score for mismatch characters is small and their algorithm runs in $O(n^2)$ time if the range of the function $W(k)$ is not functionally dependent on n.

Cross References

▶ Local Alignment (with Affine Gap Weights)

Recommended Reading

1. Altschul, S.F., Erickson, B.W.: Optimal sequence alignment using affine gap costs. Bull. Math. Biol. **48**, 603–616 (1986)
2. Benner, S.A., Cohen, M.A., Gonnet, G.H.: Empirical and structural models for insertions and deletions in the divergent evolution of proteins. J. Mol. Biol. **229**, 1065–1082 (1993)
3. Crochemore, M., Landau, G.M., Ziv-Ukelson, M.: A subquadratic sequence alignment algorithm for unrestricted scoring matrices. SIAM J. Comput. **32**(6), 1654–1673 (2003)

4. Eppstein, D.: Sequence comparison with mixed convex and concave costs. J. Algorithms **11**(1), 85–101 (1990)
5. Galil, Z., Giancarlo, R.: Speeding up dynamic programming with applications to molecular biology. Theor. Comput. Sci. **64**, 107–118 (1989)
6. Gotoh, O.: An improved algorithm for matching biological sequences. J. Mol. Biol. **162**, 705–708 (1982)
7. Gusfield, D.: Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology. Cambridge University Press, Cambridge (1997)
8. Masek, W.J., Paterson, M.S.: A fater algorithm for computing string edit distances. J. Comput. Syst. Sci. **20**, 18–31 (1980)
9. Miller, W., Myers, E.W.: Sequence comparison with concave weighting functions. Bull. Math. Biol. **50**(2), 97–120 (1988)
10. De Roberts, E., Oliver, G., Wright, C.: Homeobox genes and the vertibrate body plan, pp. 46–52. Scientific American (1990)
11. Sankoff, D., Kruskal, J.B.: Time Warps, Strings Edits, and Macromolecules: The Theory and Practice of Sequence Comparison. Addison-Wesley (1983)
12. Waterman, M.S.: Efficient sequence alignment algorithms. J. Theor. Biol. **108**, 333–337 (1984)
13. Li, W.-H., Gu, X.: The size distribution of insertions and deletions in human and rodent pseudogenes suggests the logarithmic gap penalty for sequence alignment. J. Mol. Evol. **40**, 464–473 (1995)

Local Approximation of Covering and Packing Problems

2003–2006; Kuhn, Moscibroda, Nieberg, Wattenhofer

FABIAN KUHN
Department of Computer Science, ETH Zurich,
Zurich, Switzerland

Synomyms

Distributed approximation of covering and packing problems

Problem Definition

A *local algorithm* is a distributed algorithm on a network with a running time which is independent or almost independent of the network's size or diameter. Usually, a distributed algorithm is called local if its time complexity is at most polylogarithmic in the size n of the network. Because the time needed to send information from one node of a network to another is at least proportional to the distance between the two nodes, in such an algorithm, each node's computation is based on information from nodes in a close vicinity only. Although all computations are based on local information, the network as a whole typically still has to achieve a global goal. Having local algorithms is inevitable to obtain time-efficient distributed protocols for large-scale and dynamic networks such as peer-to-peer networks or wireless ad hoc and sensor networks.

In [2,6,7], Kuhn, Moscibroda, and Wattenhofer describe upper and lower bounds on the possible trade-off between locality (time complexity) of distributed algorithms and the quality (approximation ratio) of the achievable solution for an important class of problems called covering and packing problems. Interesting covering and packing problems in the context of networks include minimum dominating set, minimum vertex cover, maximum matching, as well as certain flow maximization problems. All the results given in [2,6,7] hold for general network topologies. Interestingly, it is shown by Kuhn, Moscibroda, Nieberg, and Wattenhofer in [3,4,5] that covering and packing problems can be solved much more efficiently when assuming that the network topology has special properties which seem realistic for wireless networks.

Distributed Computation Model

In [2,3,4,5,6,7], the network is modeled as an undirected and except for [5] unweighted graph $G = (V, E)$. Two nodes $u, v \in V$ of the network are connected by an edge $(u, v) \in E$ whenever there is a direct bidirectional communication channel connecting u and v. In the following, the number of nodes and the maximal degree of G are denoted by $n = |V|$ and by Δ.

For simplicity, communication is assumed to be synchronous. That is, all nodes start an algorithm simultaneously and time is divided into rounds. In each round, every node can send an arbitrary message to each of its neighbors and perform some local computation based on the information collected in previous rounds. The *time complexity* of a synchronous distributed algorithm is the number of rounds until all nodes terminate.

Local distributed algorithms in the described synchronous model have first been considered in [8] and [9]. As an introduction to the above and similar distributed computation models, it is also recommended to read [11].

Distributed Covering and Packing Problems

A fractional covering problem (P) and its dual fractional packing problem (D), are linear programs (LPs) of the canonical forms

$$\begin{array}{llcllc} \min & \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} & & \max & \boldsymbol{b}^{\mathrm{T}}\boldsymbol{y} & \\ \text{s.t.} & A\cdot\boldsymbol{x}\ge\boldsymbol{b} & \text{(P)} & \text{s.t.} & A^{\mathrm{T}}\cdot\boldsymbol{y}\le\boldsymbol{c} & \text{(D)} \\ & \boldsymbol{x}\ge\boldsymbol{0} & & & \boldsymbol{y}\ge\boldsymbol{0} & \end{array}$$

where all a_{ij}, b_i, and c_i are non-negative. In a distributed context, finding a small (weighted) dominating set or a small (weighted) vertex cover of the network graph are the most important covering problems. A dominating set of a graph G is a subset S of its nodes such that all nodes of G either are in S or have a neighbor in S. The dominating set problem can be formulated as covering integer LP by setting A to be the adjacency matrix with 1s in the diagonal, by setting $\boldsymbol{b}$ to be a vector with all 1s and if $\boldsymbol{c}$ is the weight vector. A vertex cover is a subset of the nodes such that all edges are covered. Packing problems occur in a broad range of resource allocation problems. As an example, in [1] and [10], the problem of assigning flows to a given fixed set of paths is described. Another common packing problem is (weighted) maximum matching, the problem of finding a largest possible set of pairwise non-adjacent edges.

While computing a dominating set, vertex cover, or matching of the network graph are inherently distributed tasks, general covering and packing LPs have no immediate distributed meaning. To obtain a distributed version of these LPs, two dual LPs (P) and (D) are mapped to a bipartite network as follows. For each primal variable x_i and for each dual variable y_j, there are nodes $v_i{}^p$ and $v_j{}^d$, respectively. There is an edge between two nodes $v_i{}^p$ and $v_j{}^d$ whenever $a_{ji} \neq 0$, i. e., there is an edge if the ith variable of an LP occurs in its jth inequality.

In most real-world examples of distributed covering and packing problems, the network graph is of course not equal to the described bipartite graph. However, it is usually straightforward to simulate an algorithm which is designed for the above bipartite network on the actual network graph without affecting time and message complexities.

Bounded Independence Graphs

In [3,4,5], local approximation algorithms for covering and packing problems for graphs occuring in the context of wireless ad hoc and sensor networks are studied. Because of scale, dynamism and the scarcity of resources, these networks are a particular interesting area to apply local distributed algorithms.

Wireless networks are often modeled as *unit disk graphs* (UDGs): Nodes are assumed to be in a two-dimensional Euclidean plane and two nodes are connected by an edge iff their distance is at most 1. This certainly captures the inherent geometric nature of wireless networks. However, unit disk graphs seem much too restrictive to accurately model real wireless networks. In [3,4,5], Kuhn et. al. therefore consider two generalizations of the unit disk graph model, *bounded independent graphs* (BIGs) and *unit ball graphs* (UBGs). A BIG is a graph where all local independent sets are of bounded size. In particular, it is assumed that there is a function $I(r)$ which upper bounds the size of the largest independent set of every r-neighborhood in the graph. Note that the value of $I(r)$ is independent of n, the size of the network. If $I(r)$ is a polynomial in r, a BIG is said to be polynomially bounded. UDGs are BIGs with $I(r) \in O(r^2)$. UBGs are a natural generalization of UDGs. Given some underlying metric space (V, d) two nodes $u, v \in V$ are connected by an edge iff $d(u, v) \leq 1$. If the metric space (V, d) has constant doubling dimension[1], a UBG is a polynomially bounded BIG.

Key Results

The first algorithms to solve general distributed covering and packing LPs appear in [1,10]. In [1], it is shown that it is possible to find a solution which is within a factor of $1 + \varepsilon$ of the optimum in $O(\log^3(\rho n)/\varepsilon^3)$ rounds where ρ is the ratio between the largest and the smallest non-zero coefficient of the LPs. The result of [1] is improved and generalized in [6,7] where the following result is proven:

Theorem 1 *In k rounds, (P) and (D) can be approximated by a factor of $(\rho\Delta)^{O(1/\sqrt{k})}$ using messages of size at most $O(\log(\rho\Delta))$. An $(1 + \varepsilon)$-approximation can be found in time $O(\log^2(\rho\Delta)/\varepsilon^4)$.*

The algorithm underlying Theorem 1 needs only small messages of size $O(\log(\rho\Delta))$ and extremely simple and efficient local computations. If larger messages and more complicated (but still polynomial) local computations are allowed, it is possible to improve the result of Theorem 1:

Theorem 2 *In k rounds, LPs of the form (P) or (D) can be approximated by a factor of $O(n^{O(1/k)})$. This implies that a constant approximation can be found in time $O(\log n)$.*

Theorems 1 and 2 only give bounds on the quality of distributed solutions of covering and packing LPs. However, many of the practically relevant problems are integer versions of covering and packing LPs. Combined with simple randomized rounding schemes, the following upper bounds for dominating set, vertex cover, and matching are proven in [6,7]:

Theorem 3 *Let Δ be the maximal degree of the given network graph. In k rounds, minimum dominating set can be approximated by a factor of $O(\Delta^{O(1/\sqrt{k})} \cdot \log\Delta)$ in expectation by using messages of size $O(\Delta)$. Without bound on the message size, an expected approximation ratio of*

[1]The doubling dimension of a metric space is the logarithm of the maximal number of balls needed to cover a ball $B_r(x)$ in the metric space with balls $B_{r/2}(y)$ of half the radius.

$O(n^{O(1/k)} \cdot \log \Delta)$ can be achieved. Minimum vertex cover and maximum matching can both be approximated by a factor of $O(\Delta^{1/k})$ in k rounds.

In [2,7], it is shown that the upper bounds on the trade-offs between time complexity and approximation ratio given by Theorems 1–3 are almost optimal:

Theorem 4 *In k rounds, it is not possible to approximate minimum vertex cover better than by factors of $\Omega(\Delta^{1/k}/k)$ and $\Omega(n^{\Omega(1/k^2)}/k)$. This implies time lower bounds of $\Omega(\log \Delta / \log\log \Delta)$ and $\Omega(\sqrt{\log n / \log\log n})$ for constant or even poly-logarithmic approximation ratios. The same bounds hold for minimum dominating set, for maximum matching, as well as for the underlying LPs.*

While Theorem 4 shows that the results given by Theorems 1–3 are close to optimal for worst-case network topologies, the problems might be much simpler if restricted to networks which actually occur in reality. In fact, it is shown in [3,4,5] that the above results can indeed be improved if the network graph is assumed to be a BIG or a UBG with constant doubling dimension. In [5], the following result for UBGs is proven:

Theorem 5 *Assume that the network graph $G = (V, E)$ is a UBG with underlying metric (V, d). If (V, d) has constant doubling dimension and if all nodes know the distances to their neighbors in G up to a constant factor, it is possible to find constant approximations for minimum dominating set, minimum vertex cover, maximum matching, as well as for general LPs of the forms (P) and (D) in $O(\log^* n)$ rounds*[2].

While the algorithms underlying the results of Theorems 1 and 2 for solving covering and packing LPs are deterministic or straight-forward to be derandomized, all known efficient algorithms to solve minimum dominating set and more complicated integer covering and packing problems are randomized. Whether there are good deterministic local algorithms for dominating set and related problems is a long-standing open question . In [3], it is shown that if the network is a BIG, efficient deterministic distributed algorithms exist:

Theorem 6 *On a BIG it is possible to find constant approximations for minimum dominating set, minimum vertex cover, maximum matching, as well as for LPs of the forms (P) and (D) deterministically in $O(\log \Delta \cdot \log^* n)$ rounds.*

In [4], it is shown that on polynomially bounded BIGs, one can even go one step further and efficiently find an arbitrarily good approximation by a distributed algorithm:

[2]The log-star function $\log^* n$ is an extremely slowly increasing function which gives the number of times the logarithm has to be taken to obtain a number smaller than 1.

Theorem 7 *On a polynomially bounded BIG, there is a local approximation scheme which computes a $(1+\varepsilon)$-approximation for minimum dominating set in time $O(\log \Delta \log^*(n)/\varepsilon + 1/\varepsilon^{O(1)})$. If the network graph is a UBG with constant doubling dimension and nodes know the distances to their neighbors, a $(1+\varepsilon)$-approximation can be computed in $O(\log^*(n)/\varepsilon + 1/\varepsilon^{O(1)})$ rounds.*

Applications

The most important application environments for local algorithms are large-scale decentralized systems such as wireless ad hoc and sensor networks or peer-to-peer networks. On such networks, only local algorithms lead to scalable systems. Local algorithms are particularly well-suited if the network is dynamic and computations have to be repeated frequently.

A particular application of the minimum dominating set problem is the task of clustering the nodes of wireless ad hoc or sensor networks. Assigning each node to an adjacent node in a dominating set induces a simple clustering of the nodes. If the nodes of the dominating set (i. e., the cluster centers) are connected with each other by using additional nodes, the resulting structure can be used as a backbone for routing.

Open Problems

There are a number of open problems related to the distributed approximation of covering and packing problems in particular and to distributed approximation algorithms in general. The most obvious open problem certainly is to close the gaps between the upper bounds of Theorems 1, 2, and 3 and the lower bounds of Theorem 4. It would also be interesting to see how well other optimization problems can be approximated in a distributed manner. In particular, the distributed complexity of more general classes of linear programs remains completely open. A very intriguing unsolved problem is to determine to what extent randomization is needed to obtain time-efficient distributed algorithms. Currently, the best determinic algorithms for finding a dominating set of reasonable size and for many other problems take time $2^{O(\sqrt{\log n})}$ whereas the time complexity of the best randomized algorithms usually is at most polylogarithmic in the number of nodes.

Cross References

- ► Fractional Packing and Covering Problems
- ► Maximum Matching
- ► Randomized Rounding

Recommended Reading

1. Bartal, Y., Byers, J.W., Raz, D.: Global optimization using local information with applications to flow control. In: Proc. of the 38th IEEE Symposium on the Foundations of Computer Science (FOCS), pp. 303–312 (1997)
2. Kuhn, F., Moscibroda, T., Wattenhofer, R.: What cannot be computed locally! In: Proc. of the 23rd ACM Symp. on Principles of Distributed Computing (PODC), pp. 300–309 (2004)
3. Kuhn, F., Moscibroda, T., Nieberg, T., Wattenhofer, R.: Fast deterministic distributed maximal independent set computation on growth-bounded graphs. In: Proc. of th 19th Int. Conference on Distributed Computing (DISC), pp. 273–287 (2005)
4. Kuhn, F., Moscibroda, T., Nieberg, T., Wattenhofer, R.: Local approximation schemes for ad hoc and sensor networks. In: Proc. of the 3rd Joint Workshop on Foundations of Mobile Computing (DIALM-POMC), pp. 97–103 (2005)
5. Kuhn, F., Moscibroda, T., Wattenhofer, R.: On the locality of bounded growth. In: Proc. of the 24th ACM Symposium on Principles of Distributed Computing (PODC), pp. 60–68 (2005)
6. Kuhn, F., Wattenhofer, R.: Constant-time distributed dominating set approximation. Distrib. Comput. **17**(4), 303–310 (2005)
7. Kuhn, F., Moscibroda, T., Wattenhofer, R.: The price of being near-sighted. In: Proc. of the 17th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 980–989 (2006)
8. Linial, N.: Locality in distributed graph algorithms. SIAM J. Comput. **21**(1), 193–201 (1992)
9. Naor, M., Stockmeyer, L.: What can be computed locally? In: Proc. of the 25th Annual ACM Symposium on Theory of Computing (STOC), pp. 184–193 (1993)
10. Papadimitriou, C., Yannakakis, M.: Linear programming without the matrix. In: Proc. of the 25th ACM Symposium on Theory of Computing (STOC), pp. 121–129 (1993)
11. Peleg, D.: Distributed Computing: A Locality-Sensitive Approach. SIAM (2000)

Local Computation in Unstructured Radio Networks

2005; Moscibroda, Wattenhofer

THOMAS MOSCIBRODA
Systems & Networking Research Group, Microsoft Research, Redmond, WA, USA

Keywords and Synonyms

Maximal independent sets in radio networks; Coloring unstructured radio networks

Problem Definition

In many ways, familiar distributed computing communication models such as the *message passing model* do not describe the harsh conditions faced in wireless ad hoc and sensor networks closely enough. Ad hoc and sensor networks are multi-hop radio networks and hence, messages being transmitted may interfere with concurrent transmissions leading to collisions and packet losses. Furthermore, the fact that all nodes share the same wireless communication medium leads to an inherent broadcast nature of communication. A message sent by a node can be received by all nodes in its transmission range. These aspects of communication are modeled by the *radio network model*, e. g. [2].

Definition 1 (Radio Network Model) In the radio network model, the wireless network is modeled as a graph $G = (V, E)$. In every time-slot, a node $u \in V$ can either send or not send a message. A node v, $(u, v) \in E$, receives the message if and only *exactly one* of its neighbors has sent a message in this time-slot.

While communication primitives such as broadcast, wake-up, or gossiping, have been widely studied in the literature on radio networks (e. g., [1,2,8]), less is known about the computation of *local network coordination structures* such as clusterings or colorings. The most basic notion of a clustering in wireless networks boils down to the graph-theoretic notion of a dominating set.

Definition 2 (Minimum Dominating Set (MDS)) Given a graph $G = (V, E)$. A dominating set is a subset $S \subseteq V$ such that every node is either in S or has at least one neighbor in S. The minimum dominating set problem asks for a dominating set S of minimum cardinality.

A dominating set $S \subseteq V$ in which no two neighboring nodes are in S is a *maximal independent set (MIS)*. The distributed complexity of computing a MIS in the message passing model has been of fundamental interest to the distributed computing community for over two decades (e. g., [11,12,13]), but much less is known about the problem's complexity in radio network models.

Definition 3 (Maximal Independent Set (MIS)) Given a graph $G = (V, E)$. An independent set is a subset of pairwise non-adjacent nodes in G. A maximal independent set in G is an independent set $S \subseteq V$ such that for every node $u \notin S$, there is a node $v \in \Gamma(u)$ in S.

Another important primitive in wireless networks is the *vertex coloring* problem, because associating different colors with different time slots in a time-division multiple access (TDMA) scheme; a correct coloring corresponds to a medium access control (MAC) layer without *direct interference*, that is, no two neighboring nodes send at the same time.

Definition 4 (Minimum Vertex Coloring) Given a graph $G = (V, E)$. A correct vertex coloring for G is an assignment of a color $c(v)$ to each node $v \in V$, such that

$c(u) \neq c(v)$ any two adjacent nodes $(u, v) \in E$. A minimum vertex coloring is a correct coloring that minimizes the number of used colors.

In order to capture the especially harsh characteristics of wireless multi-hop networks immediately after their deployment, the unstructured radio network model makes additional assumptions. In particular, a new notion of asynchronous wake-up is considered, because, in a wireless, multi-hop environment, it is realistic to assume that some nodes join the network (e. g. become deployed, or switched on) later than others. Notice that this is different from the notion of asynchronous wake-up defined and studied in [8] and subsequent work, in which nodes are assumed to be "woken up" by incoming messages.

Definition 5 (Unstructured Radio Network Model) In the *unstructured radio network model*, the wireless network is modeled as a unit disk graph (UDG) $G = (V, E)$. In every time-slot, a node $u \in V$ can either send or not send a message. A node v, $(u, v) \in E$, receives the message if and only *exactly one* of its neighbors has sent a message in this time-slot. Additionally, the following assumptions are made:

- *Asynchronous wake-up:* New nodes can wake up/join in *asynchronously* at any time. Before waking-up, nodes do neither receive nor send any messages.
- *No global clock:* Nodes only have access to a local clock that starts increasing after wake-up.
- *No collision detection:* Nodes cannot distinguish between the event of a collision and no message being sent. Moreover, a sending node does not know how many (if any at all!) neighbors have received its transmission correctly.
- *Minimal global knowledge:* At the time of their wake-up, nodes have no information about their neighbors in the network and they do not whether some neighbors are already awake, executing the algorithm. However, nodes know an upper bound for the maximum number of nodes $n = |V|$.

The measure that captures the efficiency of an algorithm defined in the unstructured radio network model is its *time-complexity*. Since every node can wake up at a different time, the time-complexity of an algorithm is defined as the maximum number of time-slots between a node's wake-up and its final, irrevocable decision.

Definition 6 (Time Complexity) The *running time* T_v of a node $v \in V$ is defined as the number of time slots between v's *waking up* and the time v makes an irrevocable *final decision* on the outcome of its protocol (e. g. whether or not it joins the dominating set in a clustering algorithm, or which color to take in a coloring algorithm, etc.). The *time complexity* $T(\mathcal{Q})$ of algorithm $\mathcal{Q}$ is defined as the maximum running time over all nodes in the network, i. e., $T(\mathcal{Q}) := \max_{v \in V} T_v$.

Key Results

Naturally, algorithms for such uninitialized, chaotic networks have a different flavor compared to "traditional" algorithms that operate on a given network graph that is static and well-known to all nodes. Hence, the algorithmic difficulty of the following algorithms partly stems from the fact that since nodes wake up asynchronously and do not have access to a global clock, the different phases of the algorithm may be arbitrarily intertwined or shifted in time. Hence, while some nodes may already be in an advanced stage of the algorithm, there may be nodes that have either just woken up, or that are still in early stage. It was proven in [9] that even in single-hop networks (G is the complete graph), no efficient algorithms exist if nodes have no knowledge on n.

Theorem 1 *If nodes have no knowledge of n, every (possibly randomized) algorithm requires up to $\Omega(n/\log n)$ time slots before at least one node can send a message in single-hop networks.*

In single-hop networks, and if n is globally known, [8] presented a randomized algorithm that selects a unique leader in time $O(n \log n)$, with high probability. This result has subsequently been improved to $O(\log^2 n)$ by Jurdziński and Stachowiak [9]. The generalized wake-up problem in multi-hop radio network was first studied in [4].

The complexity of local network structures such as clusterings or colorings in unstructured multi-hop radio networks was first studied in [10]: A good approximation to the minimum dominating set problem can be computed in polylogarithmic time.

Theorem 2 *In the unstructured radio network model, an expected $O(1)$-approximation to the dominating set problem can be computed in expected time $O(\log^2 n)$. That is, every node decides whether to join the dominating set within $O(\log^2 n)$ time slots after its wake-up.*

In a subsequent paper [18], it has been shown that the running time of $O(\log^2 n)$ is sufficient even for computing the more sophisticated MIS structure. This result is asymptotically optimal because—improving on a previously known bound of $\Omega(\log^2 n/\log\log n)$ [9]—, a corresponding lower bound of $\Omega(\log^2 n)$ has been proven in [6].

Theorem 3 *With high probability, a maximal independent set (MIS) can be computed in expected time $O(\log^2 n)$*

in the unstructured radio network model. This is asymptotically optimal.

It is interesting to compare this achievable upper bound on the harsh unstructured radio network model with the best known time lower bounds in message passing models: $\Omega(\log^* n)$ in unit disk graphs [12] and $\Omega(\sqrt{\log n / \log\log n})$ in general graphs [11]. Also, a time bound of $O(\log^2 n)$ was also proven in [7] in a radio network model without asynchronous wake-up and in which nodes have a-priori knowledge about their neighborhood.

Finally, it is also possible to efficiently color the nodes of a network as shown in [17], and subsequently improved and generalized in Chap. 12 of [15].

Theorem 4 *In the unstructured radio network model, a correct coloring with at most $O(\Delta)$ colors can be computed in time $O(\Delta \log n)$ with high probability.*

Similar bounds for a model with collision detection mechanisms are proven in [3].

Applications

In wireless ad hoc and sensor networks, local network coordination structures find important applications. In particular, clusterings and colorings can help in facilitating the communication between adjacent nodes (MAC layer protocols) and between distant nodes (routing protocols), or to improve the energy efficiency of the network.

The following mentions two specific examples of applications: Based on the MIS algorithms of Theorem 3, a protocol is presented in [5], which efficiently constructs a *spanner*, i. e., a more sophisticated initial infrastructure that helps in structuring wireless multi-hop network. In [16], the same MIS algorithm is used as an ingredient for a protocol that minimizes the energy consumption of wireless sensor nodes during the *deployment phase*, a problem that has been first studied in [14].

Recommended Reading

1. Alon, N., Bar-Noy, A., Linial, N., Peleg, D.: A Lower Bound for Radio Broadcast. J. Comput. Syst. Sci. **43**, 290–298 (1991)
2. Bar-Yehuda, R., Goldreich, O., Itai, A.: On the time-complexity of broadcast in radio networks: an exponential gap between determinism randomization. In: Proc. 6th Symposium on Principles of Distributed Computing (PODC), pp. 98–108 (1987)
3. Busch, R., Magdon-Ismail, M., Sivrikaya, F., Yener, B.: Contention-Free MAC Protocols for Wireless Sensor Networks. In: Proc. 18th Annual Conference on Distributed Computing (DISC) (2004)
4. Chrobak, M., Gąsieniec, L., Kowalski, D.: The Wake-Up Problem in Multi-Hop Radio Networks. In: Proc. of the 15th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 992–1000 (2004)
5. Farach-Colton, M., Fernandes, R.J., Mosteiro, M.A.: Bootstrapping a Hop-Optimal Network in the Weak Sensor Model. In: Proc. of the 13th European Symposium on Algorithms (ESA), pp. 827–838 (2005)
6. Farach-Colton, M., Fernandes, R.J., Mosteiro, M.A.: Lower Bounds for Clear Transmissions in Radio Networks. In: Proc. of the 7th Latin American Symposium on Theoretical Informatics (LATIN), pp. 447–454 (2006)
7. Gandhi, R., Parthasarathy, S.: Distributed Algorithms for Coloring and Connected Domination in Wireless Ad Hoc Networks. In: Foundations of Software Technology and Theoretical Computer Science (FSTTCS), pp. 447–459 (2004)
8. Gąsieniec, L., Pelc, A., Peleg, D.: The Wakeup Problem in Synchronous Broadcast Systems (Extended Abstract). In: Proc. of the 19th ACM Symposium on Principles of Distributed Computing (PODC), pp. 113–121 (2000)
9. Jurdziński, T., Stachowiak, G.: Probabilistic Algorithms for the Wakeup Problem in Single-Hop Radio Networks. In: Proc. of the 13th Annual International Symposium on Algorithms and Computation (ISAAC), pp. 535–549 (2002)
10. Kuhn, F., Moscibroda, T., Wattenhofer, R.: Initializing Newly Deployed Ad Hoc and Sensor Networks. In: Proc. of the 10th Annual International Conference on Mobile Computing and Networking (MOBICOM), pp. 260–274 (2004)
11. Kuhn, F., Moscibroda, T., Wattenhofer, R.: What Cannot Be Computed Locally! In: Proceedings of 23rd Annual Symposium on Principles of Distributed Computing (PODC), pp. 300–309 (2004)
12. Linial, N.: Locality in Distributed Graph Algorithms. SIAM J. Comput. **21**(1), 193–201 (1992)
13. Luby, M.: A Simple Parallel Algorithm for the Maximal Independent Set Problem. SIAM J. Comput. **15**, 1036–1053 (1986)
14. McGlynn, M.J., Borbash, S.A.: Birthday Protocols for Low Energy Deployment and Flexible Neighborhood Discovery in Ad Hoc Wireless Networks. In: Proc. of the 2nd ACM Int. Symposium on Mobile Ad Hoc Networking & Computing (MOBIHOC), (2001)
15. Moscibroda, T.: Locality, Scheduling, and Selfishness: Algorithmic Foundations of Highly Decentralized Networks. Doctoral Thesis Nr. 16740, ETH Zurich (2006)
16. Moscibroda, T., von Rickenbach, P., Wattenhofer, R.: Analyzing the Energy-Latency Trade-off during the Deployment of Sensor Networks. In: Proc. of the 25th Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), (2006)
17. Moscibroda, T., Wattenhofer, R.: Coloring Unstructured Radio Networks. In: Proc. of the 17th ACM Symposium on Parallel Algorithms and Architectures (SPAA), pp. 39–48 (2005)
18. Moscibroda, T., Wattenhofer, R.: Maximal Independent Sets in Radio Networks. In: Proc. of the 23rd ACM Symposium on Principles of Distributed Computing (PODC), pp. 148–157 (2005)

Local Search Algorithms for *k*SAT

1999; Schöning

KAZUO IWAMA
School of Informatics, Kyoto University, Kyoto, Japan

Problem Definition

The CNF Satisfiability problem is to determine, given a CNF formula F with n variables, whether or not there exists a satisfying assignment for F. If each clause of F contains at most k literals, then F is called a k-CNF formula and the problem is called k-SAT, which is one of the most fundamental NP-complete problems. The trivial algorithm is to search 2^n 0/1-assignments for the n variables. But since [6], several algorithms which run significantly faster than this $O(2^n)$ bound have been developed. As a simple exercise, consider the following straightforward algorithm for 3-SAT, which gives us an upper bound of 1.913^n: Choose an arbitrary clause in F, say $(x_1 \vee \overline{x_2} \vee x_3)$. Then generate seven new formulas by substituting to these x_1, x_2 and x_3 all the possible values excepting $(x_1, x_2, x_3) = (0, 1, 0)$ which obviously unsatisfies F. Now one can check the satisfiability of these seven formulas and conclude that F is satisfiable iff at least one of them is satisfiable. (Let $T(n)$ denote the time complexity of this algorithm. Then one can get the recurrence $T(n) \leq 7 \times T(n-3)$ and the above bound follows.)

Key Results

In the long history of k-SAT algorithms, the one by Schöning [11] is an important breakthrough. It is a standard local search and the algorithm itself is not new (see e. g. [7]). Suppose that y is the current assignment (its initial value is selected uniformly at random). If y is a satisfying assignment, then the algorithm answers yes and terminates. Otherwise, there is at least one clause whose three literals are all false under y. Pick an arbitrary such clause and select one of the three literals in it at random. Then flip (true to false and vice versa) the value of that variable, replace y with that new assignment and then repeat the same procedure. More formally:

```
**SCH**(CNF-formula $F$, integer $I$)
  **repeat** $I$ times
    $y$ = uniformly random vector $\in \{0,1\}^n$
    $z$ = **RandomWalk**($F, y$);
    **if** $z$ satisfies $F$
      **then** output($z$); exit;
  **end**
  output('Unsatisfiable');
**RandomWalk**(CNF formula $G(x_1, x_2, \ldots, x_n)$, assignment $y$);
  $y' = y$;
  **for** $3n$ times
    **if** $y'$ satisfies $G$
      **then return** $y'$; exit;
    $C \leftarrow$ an arbitrary clause of $G$ that is not satisfied by $y'$;
    Modify $y'$ as follows:
      select one literal of $C$ uniformly at random and
      flip the assignment to this literal;
  **end**
  **return** $y'$
```

Schöning's analysis of this algorithm is very elegant. Let $d(a, b)$ denote the Hamming distance between two binary vectors (assignments) a and b. For simplicity, suppose that the formula F has only one satisfying assignment y^* and the current assignment y is far from y^* by Hamming distance d. Suppose also that the currently false clause C includes three variables, x_i, x_j and x_k. Then y and y^* must differ in at least one of these three variables. This means that if the value of x_i, x_j or x_k is flipped, then the new assignment gets closer to y^* by Hamming distance one with probability at least 1/3. Also, the new assignment gets farther by Hamming distance one with probability at most 2/3. The argument can be generalized to the case that F has multiple satisfying assignments. Now here comes the key lemma:

Lemma 1 *Let F be a satisfiable formula and y^* be a satisfying assignment for F. For each assignment y, the probability that a satisfying assignment (that may be different from y^*) is found by **RandomWalk**(F, y) is at least $(1/(k-1))^{d(y,y^*)}/p(n)$, where $p(n)$ is a polynomial in n.*

By taking the average over random initial assignments, the following theorem follows:

Theorem 2 *For any satisfiable formula F on n variables, the success probability of **RandomWalk**(F, y) is at least $(k/2(k-1))^n/p(n)$ for some polynomial p. Thus, by setting $I = (2(k-1)/k)^n \cdot p(n)$, **SCH** finds a satisfying assignment with high probability. When $k = 3$, this value of I is $O(1.334^n)$.*

Applications

The Schöning's result has been improved by a series of papers [1,3,9] based on the idea of [3]. Namely, Random Walk is combined with the (polynomial time) 2SAT algorithm, which makes it possible to choose better initial assignments. For derandomization of **SCH**, see [2]. [4] developed a nontrivial combination of **SCH** with another famous, backtrack-type algorithm by [8], resulting in the then fastest algorithm with $O(1.324^n)$ running time. The current fastest algorithm is due to [10], which is based on the same approach as [4] and runs in time $O(1.32216^n)$.

Open Problems

k-SAT is probably the most popular NP-complete problem for which numerous researchers are competing for its fastest algorithm. Thus improving its time bound is always a good research target.

Experimental Results

AI researchers have also been very active in SAT algorithms including local search, see e. g. [5].

Cross References

► Exact Algorithms for General CNF SAT
► Random Planted 3-SAT

Recommended Reading

1. Baumer, S., Schuler, R.: Improving a probabilistic 3-SAT algorithm by dynamic search and independent clause pairs. ECCC TR03-010, (2003) Also presented at SAT (2003)
2. Dantsin, E., Goerdt, A., Hirsch, E.A., Kannan, R., Kleinberg, J., Papadimitriou, C., Raghavan, P., Schöning, U.: A deterministic $(2 - 2/(k+1))^n$ algorithm for k-SAT based on local search. Theor. Comput. Sci. **289**(1), 69–83 (2002)
3. Hofmeister, T., Schöning, U., Schuler, R., Watanabe, O.: Probabilistic 3-SAT algorithm further improved. Proceedings 19th Symposium on Theoretical Aspects of Computer Science. LNCS **2285**, 193–202 (2002)
4. Iwama, K., Tamaki, S.: Improved upper bounds for 3-SA T. In: Proceedings 15th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 321–322. New Orleans, USA (2004)
5. Kautz, H., Selman, B.: Ten Challenges Redux: Recent Progress in Propositional Reasoning and Search. Proceedings 9th International Conference on Principles and Practice of Constraint Programming, pp. 1–18. Kinsale, Ireland (2003)
6. Monien, B., Speckenmeyer, E.: Solving satisfiability in less than 2^n steps. Discret. Appl. Math. **10**, 287–295 (1985)
7. Papadimitriou, C.H.: On selecting a satisfying truth assignment. Proceedings 32nd Annual Symposium on Foundations of Computer Science, pp. 163–169. San Juan, Puerto Rico (1991)
8. Paturi, R., Pudlák, P., Saks, M.E., Zane, F.: An improved exponential-time algorithm for k-SAT. Proceedings 39th Annual Symposium on Foundations of Computer Science, pp. 628–637. Palo Alto, USA (1998) Also, J. ACM **52**(3), 337–364 (2006)
9. Rolf, D.: 3-SAT $\in RTIME(O(1.32793^n))$. ECCC TR03–054. (2003)
10. Rolf, D.: Improved Bound for the PPSZ/Schöning-Algorithm for 3-SAT. J. Satisf. Boolean Model. Comput. **1**, 111–122 (2006)
11. Schöning, U.: A probabilistic algorithm for k-SAT and constraint satisfaction problems. Proceedings 40th Annual Symposium on Foundations of Computer Science, pp. 410–414. New York, USA (1999)

Local Search for K-medians and Facility Location

2001; Arya, Garg, Khandekar, Meyerson, Munagala, Pandit

KAMESH MUNAGALA
Levine Science Research Center, Duke University, Durham, NC, USA

Keywords and Synonyms

k-Medians; k-Means; k-Medioids; Facility location; Point location; Warehouse location; Clustering

Problem Definition

Clustering is a form of *unsupervised learning*, where the goal is to "learn" useful patterns in a data set $\mathcal{D}$ of size n. It can also be thought of as a data compression scheme where a large data set is represented using a smaller collection of "representatives". Such a scheme is characterized by specifying the following:

1. A *distance* metric $\mathbf{d}$ between items in the data set. This metric should satisfy the triangle inequality: $\mathbf{d}(i,j) \le \mathbf{d}(j,k) + \mathbf{d}(k,i)$ for any three items $i,j,k \in \mathcal{D}$. In addition, $\mathbf{d}(i,j) = \mathbf{d}(j,i)$ for all $i,j \in S$ and $\mathbf{d}(i,i) = 0$. Intuitively, if the distance between two items is smaller, they are more similar. The items are usually points in some high dimensional Euclidean space $\mathcal{R}^d$. The commonly used distance metrics include the Euclidean and Hamming metrics, and the cosine metric measuring the angle between the vectors representing the items.
2. The output of the clustering process is a partitioning of the data. This chapter deals with *center-based* clustering. Here, the output is a smaller set $C \subset \mathcal{R}^d$ of *centers* which best represents the input data set $S \subset \mathcal{R}^d$. It is typically the case that $|C| \ll |\mathcal{D}|$. Each item $j \in \mathcal{D}$ is *mapped to* or *approximated by* the the closest center $i \in C$, implying $\mathbf{d}(i,j) \le \mathbf{d}(i',j)$ for all $i' \in C$. Let $\sigma : \mathcal{D} \to C$ denote this mapping. This is intuitive since closer-by (similar) items will be mapped to the same center.
3. A measure of the *quality* of the clustering, which depends on the desired output. There are several commonly used measures for the quality of clustering. In each of the clustering measures described below, the goal is to choose C such that $|C| = k$ and the objective function $f(C)$ is minimized.

k-center: $f(C) = \max_{j \in \mathcal{D}} \mathbf{d}(j, \sigma(j))$.
k-median: $f(C) = \sum_{j \in \mathcal{D}} \mathbf{d}(j, \sigma(j))$.
k-means: $f(C) = \sum_{j \in \mathcal{D}} \mathbf{d}(j, \sigma(j))^2$.

All the objectives described above are NP-HARD to optimize in general metric spaces **d**, leading to the study of heuristic and approximation algorithms. In the rest of this chapter, the focus is on the k-median objective. The approximation algorithms for k-median clustering are designed for **d** being a general possibly non-Euclidean metric space. In addition, a collection $\mathcal{F}$ of possible center locations is given as input, and the set of centers C is restricted to $C \subseteq \mathcal{F}$. From the perspective of approximation, the restriction of the centers to a finite set $\mathcal{F}$ is not too restrictive – for instance, the optimal solution which is restricted to $\mathcal{F} = \mathcal{D}$ has objective value at most a factor 2 of the optimal solution which is allowed arbitrary $\mathcal{F}$. Denote $|\mathcal{D}| = n$, and $|\mathcal{F}| = m$. The running times of the heuristics designed will be polynomial in $m\,n$, and a parameter $\varepsilon > 0$. The metric space **d** is now defined over $\mathcal{D} \cup \mathcal{F}$.

A related problem to k-medians is its Lagrangean relaxation, called FACILITY LOCATION. In this problem, there is a again collection $\mathcal{F}$ of possible center locations. Each location $i \in \mathcal{F}$ has a location cost r_i. The goal is to choose a collection $C \subseteq \mathcal{F}$ of centers and construct the mapping $\sigma : \mathcal{S} \to C$ from the items to the centers such that the following function is minimized:

$$f(C) = \sum_{j\in\mathcal{D}} \mathbf{d}(j, \sigma(j)) + \sum_{i\in C} r_i \,.$$

The facility location problem effectively gets rid of the hard bound k on the number of centers in k-medians, and replaces it with the center cost term $\sum_{i\in C} r_i$ in the objective function, thereby making it a Lagrangean relaxation of the k-median problem. Note that the costs of centers can now be non-uniform.

The approximation results for both the k-median and facility location problems carry over as is to the weighted case: Each item $j \in \mathcal{D}$ is allowed to have a non-negative weight w_j. In the objective function $f(C)$, the term $\sum_{j\in\mathcal{D}} \mathbf{d}(j,\sigma(j))$ is replaced with $\sum_{j\in\mathcal{D}} w_j \cdot \mathbf{d}(j,\sigma(j))$. The weighted case is especially relevant to the FACILITY LOCATION problem where the item weights signify user demands for a resource, and the centers denote locations of the resource. In the remaining discussion, "items" and "users" are used inter-changably to denote members of the set $\mathcal{D}$.

Key Results

The method of choice for solving both the k-median and FACILITY LOCATION problems are the class of local search heuristics, which run in "local improvement" steps. At each step t, the heuristic maintains a set C_t of centers. For the k-median problem, this collection satisfies $|C_t| = k$. A local improvement step first generates a collection of new solutions $\mathcal{E}_{t+1}$ from C_t. This is done such that $|\mathcal{E}_{t+1}|$ is polynomial in the input size. For the k-median problem, in addition, each $C \in \mathcal{E}_{t+1}$ satisfies $|C| = k$. The improvement step sets $C_{t+1} = \operatorname{argmin}_{C\in\mathcal{E}_{t+1}} f(C)$. For a prespecified parameter $\varepsilon > 0$, the improvement iterations stop at the first step T where $f(C_T) \geq (1-\varepsilon) f(C_{T-1})$.

The key design issue is the specification of the start set C_0, and the construction of $\mathcal{E}_{t+1}$ from C_t. The key analysis issues are bounding the number of steps T till termination, and the quality of the final solution $f(C_T)$ against the optimal solution $f(C^*)$. The ratio $(f(C_T))/(f(C^*))$ is termed the "locality gap" of the heuristic.

Since each improvement step reduces the value of the solution by at least a factor of $(1-\varepsilon)$, the running time in terms of number of improvement steps is given by the following expression (here D is the ratio of the largest to smallest distance in the metric space over $\mathcal{D} \cup \mathcal{F}$).

$$T \leq \log_{1/(1-\varepsilon)} \left(\frac{f(C_0)}{f(C_T)}\right) \leq \frac{\log\left(\frac{f(C_0)}{f(C_T)}\right)}{\varepsilon} \leq \frac{\log(nD)}{\varepsilon}$$

which is polynomial in the input size. Each improvement step needs computation of $f(C)$ for $C \in \mathcal{E}_t$. This is polynomial in the input size since $|\mathcal{E}_t|$ is assumed to be polynomial.

k-Medians

The first local search heuristic with provable performance guarantees is presented in the work of Arya et al. [1]. The is the natural p-swap heuristic: Given the current center set C_t of size k, the set $\mathcal{E}_{t+1}$ is defined by:

$$\mathcal{E}_{t+1} = \{(C_t \setminus \mathcal{A}) \cup \mathcal{B}\,, \quad \text{where } \mathcal{A} \subseteq C_t, \mathcal{B} \subseteq \mathcal{F} \setminus C_t, |\mathcal{A}| = |\mathcal{B}| \leq p\}\,.$$

The above simply means swap at most p centers from C_t with the same number of centers from $\mathcal{F} \setminus C_t$. Recall that $|\mathcal{D}| = n$ and $|\mathcal{F}| = m$. Clearly, $|\mathcal{E}_{t+1}| \leq (k(m-k))^p \leq (km)^p$. The start set C_0 is chosen arbitrarily. The value p is a parameter which affects the running time and the approximation ratio. It is chosen to be a constant, so that $|\mathcal{E}_t|$ is polynomial in m.

Theorem 1 ([1]) *The p-swap heuristic achieves locality gap $(3 + 2/p) + \varepsilon$ in running time $O(nk(\log(nD))/\varepsilon(mk)^p)$. Furthermore, for every p there is a k-median instance where the p-swap heuristic has locality gap exactly $(3 + 2/p)$.*

Setting $p = 1/\varepsilon$, the above heuristic achieves a $3 + \varepsilon$ approximation in running time $\tilde{O}(n(mk)^{O(1/\varepsilon)})$.

Facility Location

For this problem, since there is no longer a constraint on the number of centers, the local improvement step needs to be suitably modified. There are two local search heuristics both of which yield a locality gap of $3+\varepsilon$ in polynomial time.

The "add/delete/swap" heuristic proposed by Kuehn and Hamburger [10] either adds a center to C_t, drops a center from C_t, or swaps a center in C_t with one in $\mathcal{F} \setminus C_t$. The start set C_0 is again arbitrary.

$$\mathcal{E}_{t+1} = \{(C_t \setminus \mathcal{A}) \cup \mathcal{B},\ \text{where } \mathcal{A} \subseteq C_t, \mathcal{B} \subseteq \mathcal{F} \setminus C_t, \\ |\mathcal{A}| = 0, |\mathcal{B}| = 1 \text{ or } |\mathcal{A}| = 1, |\mathcal{B}| = 0, \text{ or } |\mathcal{A}| = 1, |\mathcal{B}| = 1\}$$

Clearly, $|\mathcal{E}_{t+1}| = O(m^2)$, making the running time polynomial in the input size and $1/\varepsilon$. Korupolu, Plaxton, and Rajaraman [9] show that this heuristic achieves a locality gap of at most $5+\varepsilon$. Arya et al. [1] strengthen this analysis to show that this heuristic achieves a locality gap of $3+\varepsilon$, and that bound this is tight in the sense that there are instances where the locality gap is exactly 3.

The "add one/delete many" heuristic proposed by Charikar and Guha [2] is slightly more involved. This heuristic adds one facility and drops all facilities which become irrelevant in the new solution.

$$\mathcal{E}_{t+1} = \{(C_t \cup \{i\}) \setminus I(i),\ \text{where } i \in \mathcal{F} \setminus C_t, I(i) \subseteq C_t\}$$

The set $I(i)$ is computed as follows: Let W denote the set of items closer to i than to their assigned centers in C_t. These items are ignored from the computation of $I(i)$. For every center $s \in C_t$, let U_s denote all items which are assigned to s. If $f_s + \sum_{j \in U_s \setminus W} d_j \mathbf{d}(j,s) > \sum_{j \in U_s \setminus W} d_j \mathbf{d}(j,i)$, then it is cheaper to remove location s and reassign the items in $U_s \setminus W$ to i. In this case, s is placed in $I(i)$. Let N denote $m+n$. Computing $I(i)$ is therefore a $O(N)$ time greedy procedure, making the overall running time polynomial. Charikar and Guha [2] show the following theorem:

Theorem 2 ([2]) *The local search heuristic which attempts to add a random center $i \notin C_t$ and remove set $I(i)$, computes a $3+\varepsilon$ approximation with high probability within $T = O(N \log N(\log N + 1/\varepsilon))$ improvement steps, each with running time $O(N)$.*

Capacitated Variants

Local search heuristics are also known for capacitated variants of the k-median and facility location problems. In this variant, each possible location $i \in \mathcal{F}$ can serve at most u_i number of users. In the soft capacitated variant of facility location, some $r_i \geq 0$ copies can be opened at $i \in \mathcal{F}$ so that the facility cost is $f_i r_i$ and the number of users served is at most $r_i u_i$. The optimization goal is now to decide the value of r_i for each $i \in \mathcal{F}$ so that the assignment of users to the centers satisfies the capacity constraints at each center, and the cost of opening the centers and assigning the users is minimized. For this variant, Arya *et al.* [1] show a local search heuristic with a locality gap of $4+\varepsilon$.

In the version of facility location with hard capacities, location $i \in \mathcal{F}$ has a hard bound u_i on the number of users that can be assigned here. If all the capacities u_i are equal (uniform case), Korupolu, Plaxton, and Rajaraman [9] present an elegant local search heuristic based on solving a transshipment problem which achieves a $8+\varepsilon$ locality gap. The analysis is improved by Chudak and Williamson [4] to show a locality gap $6+\varepsilon$. The case of non-uniform capacities requires significantly new ideas – Pál, Tardos, and Wexler [14] present a network flow based local search heuristic that achieves a locality gap of $9+\varepsilon$. This bound is improved to $8+\varepsilon$ by Mahdian and Pál [12], who generalize several of the local search techniques described above in order to obtain a constant factor approximation for the variant of facility location where the facility costs are arbitrary non-decreasing functions of the demands they serve.

Related Algorithmic Techniques

Both the k-median and facility location problems have a rich history of approximation results. Since the study of uncapacitated facility location was initiated by Cornuejols, Nemhauser, and Wolsey [5], who presented a natural linear programming (LP) relaxation for this problem, several constant-factor approximations have been designed via several techniques, ranging from rounding of the LP solution [11,15], local search [2,9], the primal-dual schema [7], and dual fitting [6]. For the k-median problem, the first constant factor approximation [3] of $6\frac{2}{3}$ was obtained by rounding the natural LP relaxation via a generalization of the filtering technique in [11]. This result was subsequently improved to a 4 approximation by Lagrangean relaxation and the primal-dual schema [2,7], and finally to a $(3+\varepsilon)$ approximation via local search [1].

Applications

The facility location problem has been widely studied in operations research [5,10], and forms a fundamental primitive for several resource location problems. The k-medians and k-means metrics are widely used in clustering, or unsupervised learning. For clustering applications, several heuristic improvements to the basic local search framework have been proposed: k-Medioids [8] selects a random input point and replaces it with one of the existing centers if there is an improvement; the CLARA [8] implementation of k-Medioids chooses the centers from a random sample of the input points to speed up the computation; the CLARANS [13] heuristic draws a fresh random sample of feasible centers before each improvement step to further improve the efficiency.

Cross References

► Facility Location

Recommended Reading

1. Arya, V., Garg, N., Khandekar, R., Meyerson, A., Munagala, K., Pandit, V.: Local search heuristics for k-median and facility location problems. SIAM J. Comput. **33**(3), 544–562 (2004)
2. Charikar, M., Guha, S.: Improved combinatorial algorithms for facility location problems. SIAM J. Comput. **34**(4), 803–824 (2005)
3. Charikar, M., Guha, S., Tardos, É., Shmoys, D.B.: A constant-factor approximation algorithm for the k-median problem (extended abstract). In: STOC '99: Proceedings of the thirty-first annual ACM symposium on Theory of computing, pp. 1–10. Atlanta, May 1-4 1999
4. Chudak, F.A., Williamson, D.P.: Improved approximation algorithms for capacitated facility location problems. Math. Program. **102**(2), 207–222 (2005)
5. Cornuejols, G., Nemhauser, G.L., Wolsey, L.A.: The uncapacitated facility location problem. In: Discrete Location Theory, pp. 119–171. Wiley, New York (1990)
6. Jain, K., Mahdian, M., Markakis, E., Saberi, A., Vazirani, V.V.: Greedy facility location algorithms analyzed using dual fitting with factor-revealing LP. J. ACM **50**(6), 795–824 (2003)
7. Jain, K., Vazirani, V.V.: Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and lagrangian relaxation. J. ACM **48**(2), 274–296 (2001)
8. Kaufman, L., Rousseeuw, P.J.: Finding Groups in Data: An Introduction to Cluster Analysis. Wiley, New York (1990)
9. Korupolu, M.R., Plaxton, C.G., Rajaraman, R.: Analysis of a local search heuristic for facility location problems. In: SODA '98: Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms, pp. 1–10. San Francisco, USA; 25–26 January 1998
10. Kuehn, A.A., Hamburger, M.J.: A heuristic program for locating warehouses. Management Sci. **9**(4), 643–666 (1963)
11. Lin, J.-H., Vitter, J.S.: ε-approximations with minimum packing constraint violation (extended abstract). In: STOC '92: Proceedings of the twenty-fourth annual ACM symposium on Theory of computing, pp. 771–782. Victoria (1992)
12. Mahdian, M., Pál, M.: Universal facility location. In: European Symposium on Algorithms, pp. 409–421. Budapest, Hungary, September 16–19 2003
13. Ng, R.T., Han, J.: Efficient and effective clustering methods for spatial data mining. In: Proc. Symp. on Very Large Data Bases (VLDB), pp. 144–155. Santiago de Chile, 12–15 September 1994
14. Pál, M., Tardos, É., Wexler, T.: Facility location with nonuniform hard capacities. In: Proceedings of the 42nd Annual Symposium on Foundations of Computer Science, pp. 329–338. Las Vegas, 14–17 October 2001
15. Shmoys, D.B., Tardos, É., and Aardal, K.: Approximation algorithms for facility location problems. In: Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing, pp. 265–274. El Paso, 4–6 May 1997

Location-Based Routing

► Geographic Routing
► Routing in Geometric Networks

Lower Bounds for Dynamic Connectivity

2004; Pătraşcu, Demaine

MIHAI PĂTRAŞCU
CSAIL, MIT, Cambridge, MA, USA

Keywords and Synonyms

Dynamic trees

Problem Definition

The dynamic connectivity problem requests maintenance of a graph G subject to the following operations:

insert(u, v): insert an undirected edge (u, v) into the graph.
delete(u, v): delete the edge (u, v) from the graph.
connected(u, v): test whether u and v lie in the same connected component.

Let m be an upper bound on the number of edges in the graph. This entry discusses cell-probe lower bounds for this problem. Let t_u be the complexity of insert and delete and t_q the complexity of query.

The Partial-Sums Problem

Lower bounds for dynamic connectivity are intimately related to lower bounds for another classic problem: maintaining partial sums. Formally, the problem asks one to maintain an array $A[1..n]$ subject to the following operations:

update(k, Δ): let $A[k] \leftarrow \Delta$.
sum(k): returns the partial sum $\sum_{i=1}^{k} A[i]$.
testsum(k, σ): returns a boolean value indicating whether sum$(k) = \sigma$.

To specify the problem completely, let elements $A[i]$ come from an arbitrary group G containing at least 2^δ elements. In the cell-probe model with b-bit cells, let t_u^Σ be the complexity of `update` and t_q^Σ the complexity of `testsum` (which is also a lower bound on `sum`).

The tradeoffs between t_u^Σ and t_q^Σ are well understood for all values of b and δ. However, this entry only considers lower bounds under the standard assumptions that $b = \Omega(\lg n)$ and $t_u \geq t_q$. It is standard to assume $b = \Omega(\lg n)$ for upper bounds in the RAM model; this assumption also means that the lower bound applies to the pointer machine. Then, Pătraşcu and Demaine [6] prove:

Theorem 1 *The complexity of the partial-sums problems satisfies:* $t_q^\Sigma \cdot \lg(t_u^\Sigma / t_q^\Sigma) = \Omega(\delta/b \cdot \lg n)$.

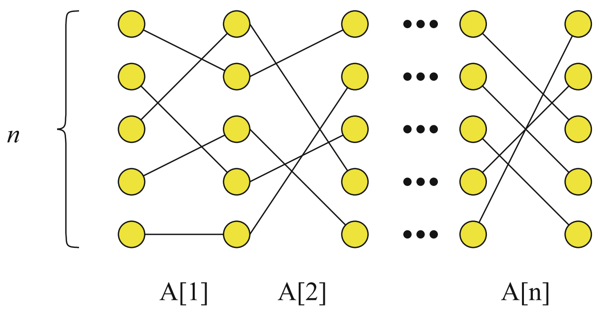
Observe that this matches the textbook upper bound using augmented trees. One can build a balanced binary tree over $A[1], \ldots, A[n]$ and store in every internal node the sum of its subtree. Then, updates and queries touch $O(\lg n)$ nodes (and spend $O(\lceil \delta/b \rceil)$ time in each one due to the size of the group). To decrease the query time, one can use a B-tree.

Relation to Dynamic Connectivity

We now clarify how lower bounds for maintaining partial sums imply lower bounds for dynamic connectivity. Consider the partial-sums problem over the group $G = S_n$, i. e., the permutation group on n elements. Note that $\delta = \lg(n!) = \Omega(n \lg n)$. It is standard to set $b = \Theta(\lg n)$, as this is the natural word size used by dynamic connectivity upper bounds. This implies $t_q^\Sigma \lg(t_u^\Sigma / t_q^\Sigma) = \Omega(n \lg n)$.

The lower bound follows from implementing the partial-sums operations using dynamic connectivity operations. Refer to Fig. 1. The vertices of the graph form an integer grid of size $n \times n$. Each vertex is incident to at most two edges, one edge connecting to a vertex in the previous column and one edge connecting to a vertex in the next column. Point (x, y_1) in the grid is connected to point $(x+1, A[x](y_1))$, i. e.,the edges between two adjacent columns describe the corresponding permutation from the partial-sums vector.

To implement `update`(x, π), all the edges between column x and $x+1$ are first deleted and then new edges are inserted according to π. This gives $t_u^\Sigma = O(2n \cdot t_u)$. To implement `testsum`$(x, \pi)$, one can use n `connected` queries between the pairs of points $(1, y) \rightsquigarrow (x+1, \pi(y))$. Then, $t_q^\Sigma = O(n \cdot t_q)$. Observe that the `sum` query cannot be implemented as easily. Dynamic connectivity is the main motivation to study the `testsum` query.



Lower Bounds for Dynamic Connectivity, Figure 1
Constructing an instance of dynamic connectivity that mimics the partial-sums problem

The lower bound of Theorem 1 translates into $nt_q \cdot \lg(2nt_u/nt_q) = \Omega(n \lg n)$; hence $t_q \lg(t_u/t_q) = \Omega(\lg n)$. Note that this lower bound implies $\max\{t_u, t_q\} = \Omega(\lg n)$. The best known upper bound (using amortization and randomization) is $O(\lg n (\lg\lg n)^3)$ [9]. For any $t_u = \Omega(\lg n (\lg\lg n)^3)$, the lower bound tradeoff is known to be tight. Note that the graph in the lower bound is always a disjoint union of paths. This implies optimal lower bounds for two important special cases: dynamic trees [8] and dynamic connectivity in plane graphs [2].

Key Results

Understanding Hierarchies

Epochs To describe the techniques involved in the lower bounds, first consider the `sum` query and assume $\delta = b$. In 1989, Fredman and Saks [3] initiated the study of dynamic cell-probe lower bounds, essentially showing a lower bound of $t_q^\Sigma \lg t_u^\Sigma = \Omega(\lg n)$. Note that this implies $\max\{t_q^\Sigma, t_u^\Sigma\} = \Omega(\lg n / \lg\lg n)$.

At an intuitive level, their argument proceeded as follows. The hard instance will have n random updates, followed by one random query. Leave $r \geq 2$ to be determined. Looking *back* in time from the query, one groups the updates into exponentially growing epochs: the latest r updates are epoch 1, the earlier r^2 updates are epoch 2, etc. Note that epoch numbers increase going back in time, and there are $O(\log_r n)$ epochs in total.

For some epoch i, consider revealing to the query all updates performed in all epochs different from i. Then, the query reduces to a partial-sums query among the updates in epoch i. Unless the query is to an index below the minimum index updated in epoch i, the answer to the query is still uniformly random, i. e., has δ bits of entropy. Furthermore, even if one is given, say, $r^i \delta/100$ bits of information about epoch i, the answer still has $\Omega(\delta)$ bits of entropy on average. This is because the query and updates in epoch i are uniformly random, so the query can ask for any partial sum of these updates, uniformly at random. Each of the r^i partial sums is an independent random variable of entropy δ.

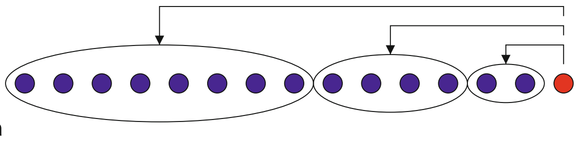
Now one can ask how much information is available to the query. At the time of the query, let each cell be associated with the epoch during which it was last written. Choosing an epoch i uniformly at random, one can make the following intuitive argument:

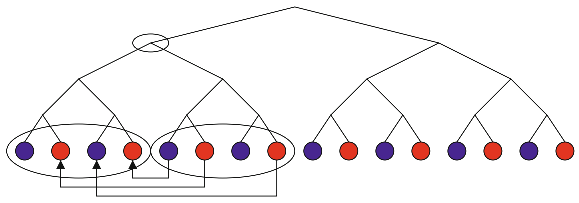
1. No cells written by epochs $i+1, i+2, \ldots$ can contain information about epoch i, as they were written in the past.
2. In epochs $1, \ldots, i-1$, a number of $bt_u^\Sigma \cdot \sum_{j=1}^{i-1} r^j \le bt_u^\Sigma \cdot 2r^{i-1}$ bits were written. This is less than $r^i\delta/100$ bits of information for $r > 200t_u^\Sigma$ (recall the assumption $\delta = b$). By the above, this implies the query answer still has $\Omega(\delta)$ bits of entropy.
3. Since i is uniformly random among $\Theta(\log_r n)$ epochs, the query makes an expected $O(t_q^\Sigma/\log_r n)$ probes to cells from epoch i. All queries that make no cell probes to epoch i have a fixed answer (entropy 0), and all other queries have answers of entropy $\le \delta$. Since an average query has entropy $\Omega(\delta)$, a query must probe a cell from epoch i with constant probability. That means $t_q^\Sigma/\log_r n = \Omega(1)$, and $\sum = \Omega(\log_r n) = \Omega(\lg n/\lg t_u^\Sigma)$.

One should appreciate the duality between the proof technique and the natural upper bounds based on a hierarchy. Consider an upper bound based on a tree of degree r. The last r random updates (epoch 1) are likely to be uniformly spread in the array. This means the updates touch different children of the root. Similarly, the r^2 updates in epoch 2 are likely to touch every node on level 2 of the tree, and so on. Now, the lower bound argues that the query needs to traverse a root-to-leaf path, probing a node on every level of the tree (this is equivalent to one cell from every epoch).

Time Hierarchies Despite considerable refinement to the lower bound techniques, the lower bound of $\Omega(\lg n/\lg\lg n)$ was not improved until 2004. Then, Pătraşcu and Demaine [6] showed an optimal bound of $t_q^\Sigma \lg(t_u^\Sigma/t_q^\Sigma) = \Omega(\lg n)$, implying $\max\{t_u^\Sigma, t_q^\Sigma\} = \Omega(\lg n)$. For simplicity, the discussion below disregards the tradeoff and just sketches the $\Omega(\lg n)$ lower bound.

Pătraşcu and Demaine's [6] counting technique is rather different from the epoch technique; refer to Fig. 2. The hard instance is a sequence of n operations alternating between updates and queries. They consider a balanced binary tree over the time axis, with every leaf being an operation. Now for every node of the tree, they propose to count the number of cell probes made in the right subtree to a cell written in the left subtree. Every probe is counted exactly once, for the lowest common ancestor of the read and write times.





Lower Bounds for Dynamic Connectivity, Figure 2
Analysis of cell probes in the a epoch-based and b time-hierarchy techniques

Now focus on two sibling subtrees, each containing k operations. The $k/2$ updates in the left subtree, and the $k/2$ queries in the right subtree, are expected to interleave in index space. Thus, the queries in the right subtree ask for $\Omega(k)$ different partial sums of the updates in the left subtree. Thus, the right subtree "needs" $\Omega(k\delta)$ bits of information about the left subtree, and this information can only come from cells written in the left subtree and read in the right one. This implies a lower bound of $\Omega(k)$ probes, associated with the parent of the sibling subtrees. This bound is linear in the number of leaves, so summing up over the tree, one obtains a total $\Omega(n \lg n)$ lower bound, or $\Omega(\lg n)$ cost per operation.

An Optimal Epoch Construction Rather surprisingly, Pătraşcu and Tarniţă [7] managed to reprove the optimal tradeoff of Theorem 1 with minimal modifications to the epoch argument. In the old epoch argument, the information revealed by epochs $1, \ldots, i-1$ about epoch i was bounded by the number of cells written in these epochs. The key idea is that an equally good bound is the number of cells read during epochs $1, \ldots, i-1$ and written during epoch i.

In principle, all cell reads from epoch $i-1$ could read data from epoch i, making these two bounds identical. However, one can randomize the epoch construction by inserting the query after an unpredictable number of updates. This randomization "smooths" out the distribution of epochs from which cells are read, i. e., a query reads $O(t_q^\Sigma / \log_r n)$ cells from every epoch, in expectation over the randomness in the epoch construction. Then, the $O(r^{i-1})$ updates in epochs $1, \dots, i-1$ only read $O(r^{i-1} \cdot t_u^\Sigma / \log_r n)$ cells from epoch i. This is not enough information if $r \gg t_u^\Sigma / \log_r n = \Theta(t_u^\Sigma / t_q^\Sigma)$, which implies $t_q^\Sigma = \Omega(\log_r n) = \Omega(\lg n / \lg(t_u^\Sigma / t_q^\Sigma))$.

Technical Difficulties

Nondeterminism The lower bounds sketched above are based on the fact that the `sum` query needs to output $\Omega(\delta)$ bits of information about every query. If dealing with the *decision* `testsum` query, an argument based on output entropy can no longer work.

The most successful idea for decision queries has been to convert them to queries with nonboolean output, in an extended cell-probe model that allows nondeterminism. In this model, the query algorithm is allowed to spawn an arbitrary number of computation threads. Each thread can make t_q cell probes, after with it must either terminate with a 'reject' answer, or return an answer to the query. All nonrejecting threads must return the same output. In this model, a query with arbitrary output is equivalent to a decision query, because one can just nondeterministically guess the answer, and then verify it.

By the above, the challenge is to prove good lower bounds for `sum` even in the nondeterminstic model. Nondeterminism shakes our view that when analyzing epoch i, only cell probes to epoch i matter. The trouble is that the query may not know *which* of its probes are actually to epoch i. A probe that reads a cell from a previous epoch provides at least some information about epoch i: no update in the epoch decided to overwrite the cell. Earlier this was not a problem because the goal was only to rule out the case that there are *zero* probes to epoch i. Now, however, different threads can probe any cell in memory, and one cannot determine which threads actually avoid probing anything in epoch i. In other words, there is a covert communication channel between epoch i and the query in which the epoch can use the choice of which cells to write in order to communicate information to the query.

There are two main strategies for handling nondeterministic query algorithms. Husfeldt and Rauhe [4] give a proof based on some interesting observations about the combinatorics of nondeterministic queries. Pătraşcu and Demaine [6] use the power of nondeterminism itself to output a small certificate that rules out useless cell probes. The latter result implies the optimal lower bound of Theorem 1 for `testsum` and, thus, the logarithmic lower bound for dynamic connectivity.

Alternative Histories The framework described above relies on fixing all updates in epochs different from i to an average value and arguing that the query answer still has a lot of variability, depending on updates in epoch i. This is true for aggregation problems but not for search problems. If a searched item is found with equal probability in any epoch, then fixing all other epochs renders epoch i irrelevant with probability $1 - 1/(\log_r n)$.

Alstrup et al. [1] propose a very interesting refinement to the technique, proving $\Omega(\lg n / \lg \lg n)$ lower bounds for an impressive collection of search problems. Intuitively, their idea is to consider $O(\log_r n)$ alternative histories of updates, chosen independently at random. Epoch i is relevant in at least one of the histories with constant probability. On the other hand, even if one knows what epochs 1 through $i-1$ learned about epoch i in *all histories*, answering a random query is still hard.

Bit-Probe Complexity Intuitively, if the word size is $b = 1$, the lower bound for connectivity should be roughly $\Omega(\lg^2 n)$, because a query needs $\Omega(\lg n)$ bits from every epoch. However, ruling out anything except zero probes to an epoch turns out to be difficult, for the same reason that the nondeterministic case is difficult. Without giving a very satisfactory understanding of this issue, Pătraşcu and Tarniţă [7] use a large bag of tricks to show an $\Omega((\lg n / \lg \lg n)^2)$ lower bound for dynamic connectivity. Furthermore, they consider the partial-sums problem in $\mathbb{Z}_2$ and show an $\Omega(\lg n / \lg \lg \lg n)$ lower bound, which is a triply-logarithmic factor away from the upper bound!

Applications

The lower bound discussed here extends by easy reductions to virtually all natural fully dynamic graph problems [6].

Open Problems

By far, the most important challenge for future research is to obtain a lower bound of $\omega(\lg n)$ per operation for some dynamic data structure in the cell-probe model with word size $\Theta(\lg n)$. Miltersen [5] specifies a set of technical conditions for what qualifies as a solution to such a challenge. In particular, the problem should be a *dynamic language membership* problem.

For the partial-sums problem, though `sum` is perfectly understood, `testsum` still lacks tight bounds for certain ranges of parameters [6]. In addition, obtaining tight bounds in the bit-probe model for partial sums in $\mathbb{Z}_2$ appears to be rather challenging.

Recommended Reading

1. Alstrup, S., Husfeldt, T., Rauhe, T.: Marked ancestor problems. In: Proc. 39th IEEE Symposium on Foundations of Computer Science (FOCS), 1998, pp. 534–543
2. Eppstein, D., Italiano, G.F., Tamassia, R., Tarjan, R.E., Westbrook, J.R., Yung, M.: Maintenance of a minimum spanning forest in a dynamic planar graph. J. Algorithms **13**, 33–54 (1992). See also SODA'90
3. Fredman, M.L., Saks, M.E.: The cell probe complexity of dynamic data structures. In: Proc. 21st ACM Symposium on Theory of Computing (STOC), 1989, pp. 345–354
4. Husfeldt, T., Rauhe, T.: New lower bound techniques for dynamic partial sums and related problems. SIAM J. Comput. **32**, 736–753 (2003). See also ICALP'98
5. Miltersen, P.B.: Cell probe complexity - a survey. In: 19th Conference on the Foundations of Software Technology and Theoretical Computer Science (FSTTCS), 1999 (Advances in Data Structures Workshop)
6. Pătraşcu, M. and Demaine, E.D.: Logarithmic lower bounds in the cell-probe model. SIAM J. Comput. **35**, 932–963 (2006). See also SODA'04 and STOC'04
7. Pătraşcu, M., Tarniţă, C.: On dynamic bit-probe complexity. Theor. Comput. Sci. **380**, 127–142 (2007). See also ICALP'05
8. Sleator, D.D., Tarjan, R.E.: A data structure for dynamic trees. J. Comput. Syst. Sci. **26**, 362–391 (1983). See also STOC'81
9. Thorup, M.: Near-optimal fully-dynamic graph connectivity. In: Proc. 32nd ACM Symposium on Theory of Computing (STOC), 2000, pp. 343–350

Low Stretch Spanning Trees

2005; Elkin, Emek, Spielman, Teng

Michael Elkin
Department of Computer Science, Ben-Gurion University, Beer-Sheva, Israel

Keywords and Synonyms

Spanning trees with low average stretch

Problem Definition

Consider a weighted connected multigraph $G = (V, E, \omega)$, where ω is a function from the edge set E of G into the set of positive reals. For a path P in G, the *weight* of P is the sum of weights of edges that belong to the path P. For a pair of vertices $u, v \in V$, the *distance* between them in G is the minimum weight of a path connecting u and v in G. For a spanning tree T of G, the stretch of an edge $(u, v) \in E$ is defined by

$$stretch_T(u, v) = \frac{dist_T(u, v)}{dist_G(u, v)} ,$$

and the average stretch over all edges of E is

$$avestr(G, T) = \frac{1}{|E|} \sum_{(u,v)\in E} stretch_T(u, v) .$$

The average stretch of a multigraph $G = (V, E, \omega)$ is defined as the smallest average stretch of a spanning tree T of G, $avestr(G, T)$. The average stretch of a positive integer n, $avestr(n)$, is the maximum average stretch of an n-vertex multigraph G. The problem is to analyze the asymptotic behavior of the function $avestr(n)$.

A closely related (dual) problem is to construct a probability distribution $\mathcal{D}$ of spanning trees for G, so that

$$expstr(G, \mathcal{D}) = \max_{e=(u,v)\in E} \mathbb{E}_{T\in\mathcal{D}}(stretch_T(u, v))$$

is small as possible. Analogously, $expstr(G) = \min_{\mathcal{D}}\{expstr(G, \mathcal{D})\}$, where the minimum is over all distributions $\mathcal{D}$ of spanning trees of G, and $expstr(n) = \max_G\{expstr(G)\}$, where the maximum is over all n-vertex multigraphs.

By viewing the problem as a 2-player zero-sum game between a tree player that aims to minimize the payoff, and an edge player that aims to maximize it, it is easy to see that for every positive integer n, $avestr(n) = expstr(n)$ [2]. The probabilistic version of the problem is, however, particularly convenient for many applications.

Key Results

The problem was studied since sixties [8,13,15,16]. A major progress in its study was achieved by Alon et al. [2], who showed that

$$\Omega(\log n) = avestr(n) = expstr(n)$$
$$= exp(O(\sqrt{\log n \cdot \log\log n})) .$$

Elkin et al. [9] improved the upper bound and showed that

$$avestr(n) = expstr(n) = O(\log^2 n \cdot \log\log n) .$$

Applications

One application of low stretch spanning trees is for solving symmetric diagonally dominant linear systems of equations. Boman and Hendrickson [5] were the first to discover the surprising relationship between these two seemingly unrelated problems. They applied the spanning trees of [2] to design solvers that run in time $m^{3/2} 2^{O(\sqrt{\log n \log\log n})} \log(1/\epsilon)$. Spielman and Teng [14] improved their results by showing how to use the spanning trees of [2] to solve diagonally-dominant linear systems in time

$$m 2^{O(\sqrt{\log n \log\log n})} \log(1/\epsilon).$$

By applying the low-stretch spanning trees developed in [9], the time for solving these linear systems reduces to

$$m \log^{O(1)} n \log(1/\epsilon),$$

and to $O(n(\log n \log\log n)^2 \log(1/\epsilon))$ when the systems are planar. Applying a recent reduction of Boman, Hendrickson and Vavasis [6], one obtains a $O(n(\log n \log\log n)^2 \log(1/\epsilon))$ time algorithm for solving the linear systems that arise when applying the finite element method to solve two-dimensional elliptic partial differential equations.

Recently Chekuri et al. [7] used low stretch spanning trees to devise an approximation algorithm for non-uniform buy-at-bulk network design problem. Their algorithm provides a first polylogarithmic approximation guarantee for this problem.

In another recent work Abraham et al. [1] use a techinique of star-decomposition introduced by Elkin et al. [9] to construct embeddings with a constant average stretch, where the average is over all *pairs of vertices*, rather than over all edges. The result of Abraham et al. [1] was, in turn, already used in a yet more recent work of Elkin et al. [10] on fundamental circuits.

Open Problems

The most evident open problem is to close the gap between the upper bound of $O(\log^2 n \log\log n)$ and the lower bound of $\Omega(\log n)$ on $avestr(n)$. Another intriguing subject is the study of low stretch spanning trees for various restricted families of graphs. Progress in this direction was recently achieved by Emek and Peleg [11] that constructed low stretch spanning trees with average stretch $O(\log n)$ for unweighted series-parallel graphs. Discovering other applications of low stretch spanning trees is another promising venue of study.

Finally, there is a closely related relaxed notion of low stretch *Steiner* or *Bartal* trees. Unlike a spanning tree, a Steiner tree does not have to be a subgraph of the original graph, but rather is allowed to use edges and vertices that were not present in the original graph. It is, however, required that the distances in the Steiner tree will be no smaller than the distances in the original graph. Low stretch Steiner trees were extensively studied [3,4,12]. Fakcharoenphol et al. [12] devised a construction of low stretch Steiner trees with an average stretch of $O(\log n)$. It is currently unknown whether the techniques used in the study of low stretch Steiner trees can help improving the bounds for the low stretch spanning trees.

Cross References

► Approximating Metric Spaces by Tree Metrics

Recommended Reading

1. Abraham, I., Bartal, Y., Neiman, O.: Embedding Metrics into Ultrametrics and Graphs into Spanning Trees with Constant Average Distortion. In: Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms, New Orleans, January 2007
2. Alon, N., Karp, R.M., Peleg, D., West, D.: A graph-theoretic gane and its application to the *k*-server problem. SIAM J. Comput. **24**(1), 78–100 (1995). Also available Technical Report TR-91-066, ICSI, Berkeley (1991)
3. Bartal, Y.: Probabilistic approximation of metric spaces and its algorithmic applications. In: Proceedings of the 37th Annual Symposium on Foundations of Computer Science, Berlington, Oct. 1996 pp. 184–193
4. Bartal, Y.: On approximating arbitrary metrices by tree metrics. In: Proceedings of the 30th annual ACM symposium on Theory of computing, Dallas, 23–26 May 1998, pp. 161–168
5. Boman, E., Hendrickson, B.: On spanning tree preconditioners. Manuscript, Sandia National Lab. (2001)
6. Boman, E., Hendrickson, B., Vavasis, S.: Solving elliptic finite element systems in near-linear time with suppost preconditioners. Manuscript, Sandia National Lab. and Cornell, http://arXiv.org/abs/cs/0407022 Accessed 9 July 2004
7. Chekuri, C., Hagiahayi, M.T., Kortsarz, G., Salavatipour, M.: Approximation Algorithms for Non-Uniform Buy-at-Bulk Network Design. In: Proceedings of the 47th Annual Symp. on Foundations of Computer Science, Berkeley, Oct. 2006, pp. 677–686
8. Deo, N., Prabhu, G.M., Krishnamoorthy, M.S.: Algorithms for generating fundamental cycles in a graph. ACM Trans. Math. Softw. **8**, 26–42 (1982)
9. Elkin, M., Emek, Y., Spielman, D., Teng, S.-H.: Lower-Stretch Spanning Trees. In: Proc. of the 37th Annual ACM Symp. on Theory of Computing, STOC'05, Baltimore, May 2005, pp. 494–503
10. Elkin, M., Liebchen, C., Rizzi, R.: New Length Bounds for Cycle Bases. Inf. Proc. Lett. **104**(5), 186–193 (2007)
11. Emek, Y., Peleg, D.: A tight upper bound on the probabilistic embedding of series-parallel graphs. In: Proc. of Symp. on Discr. Algorithms, SODA'06, Miami, Jan. 2006, pp. 1045–1053
12. Fakcharoenphol, J., Rao, S., Talwar, K.: A tight bound on approximating arbitrary metrics by tree metrics. In: Proceedings of the 35th annual ACM symposium on Theory of Computing, San Diego, June 2003, pp. 448–455
13. Horton, J.D.: A Polynomial-time algorithm to find the shortest cycle basis of a graph. SIAM J. Comput. **16**(2), 358–366 (1987)
14. Spielman, D., Teng, S.-H.: Nearly-linear time algorithm for graph partitioning, graph sparsification, and solving linear systems. In: Proc. of the 36th Annual ACM Symp. on Theory of Computing, STOC'04, Chicago. USA, June 2004, pp. 81–90
15. Stepanec, G.F.: Basis systems of vector cycles with extremal properties in graphs. Uspekhi Mat. Nauk **19**, 171–175 (1964). (In Russian)
16. Zykov, A.A.: Theory of Finite Graphs. Nauka, Novosibirsk (1969). (In Russian)

LP Decoding

2002 and later; Feldman, Karger, Wainwright

JONATHAN FELDMAN
Google, Inc., New York, NY, USA

Keywords and Synonyms

LP decoding; Error-correcting codes; Low-density parity-check codes; LDPC codes; Pseudocodewords; Belief propagation

Problem Definition

Error-correcting codes are fundamental tools used to transmit digital information over unreliable channels. Their study goes back to the work of Hamming and Shannon, who used them as the basis for the field of information theory. The problem of decoding the original information up to the full error-correcting potential of the system is often very complex, especially for modern codes that approach the theoretical limits of the communication channel.

LP decoding [4,5,8] refers to the application of *linear programming* (LP) *relaxation* to the problem of decoding an error-correcting code. Linear programming relaxation is a standard technique in approximation algorithms and operations research, and is central to the study of efficient algorithms to find good (albeit suboptimal) solutions to very difficult optimization problems [13]. LP decoders have tight combinatorial characterizations of decoding success that can be used to analyze error-correcting performance.

The codes for which LP decoding has received the most attention are *low-density parity-check* (LDPC) codes [9], due to their excellent error-correcting performance. The LP decoder is particularly attractive for analysis of these codes because the standard message-passing algorithms such as *belief propagation* (see [15]) used for decoding are often difficult to analyze, and indeed the performance of LP decoding is closely tied to these methods.

Error-Correcting Codes and Maximum-Likelihood Decoding

This section begins with a very brief overview of error-correcting codes, sufficient for formulating the LP decoder. Some terms are not defined for space reasons; for a full treatment of error-correcting codes in context, the reader is referred to textbooks on the subject (e. g., [11]).

A *binary error-correcting code* is a subset $C \subseteq \{0,1\}^n$. The *rate* of the code C is $r = \log(|C|)/n$. A *linear* binary code is a linear subspace of $\{0,1\}^n$. A *codeword* is a vector $y \in C$. Note that 0^n is always a codeword of a linear code, a fact that will be useful later. When the code is used for communication, a codeword $\dot{y} \in C$ is transmitted over a *noisy channel*, resulting in some *received word* $\hat{y} \in \Sigma^n$, where Σ is some alphabet that depends on the channel model. Generally in LP decoding a *memoryless, symmetric* channel is assumed. One common such channel is the *binary symmetric channel (BSC) with parameter* p, which will be referred to as BSC_p, where $0 < p < 1/2$. In the BSC_p, the alphabet is $\Sigma = \{0,1\}$, and for each i, the received symbol $\hat{y}_i$ is equal to $\dot{y}_i$ with probability p, and $\hat{y}_i = 1 - \dot{y}_i$ otherwise. Although LP decoding works with more general channels, this chapter will focus on the BSC_p.

The *maximum-likelihood (ML) decoding problem* is the following: given a received word $\hat{y} \in \{0,1\}^n$, find the codeword $y^* \in C$ that is most likely to have been sent over the channel. Defining the vector $\gamma \in \{-1,+1\}^n$ where $\gamma_i = 1 - 2\hat{y}_i$, it is easy to show:

$$y^* = \arg\min_{y \in C} \sum_i \gamma_i y_i \,. \tag{1}$$

The complexity of the ML decoding problem depends heavily on the code being used. For simple codes such as a *repetition code* $C = \{0^n, 1^n\}$, the task is easy. For more complex (and higher-rate) codes such as LDPC codes, ML decoding is NP-hard [1].

LP Decoding

Since ML decoding can be very hard in general, one turns to sub-optimal solutions that can be found efficiently. LP decoding, instead of trying to solve (1), relaxes the constraint $y \in C$, and instead requires that $y \in \mathcal{P}$ for some succinctly describable linear polytope $\mathcal{P} \subseteq [0,1]^n$, resulting in the following linear program:

$$y_{\text{LP}} = \arg\min_{y \in \mathcal{P}} \sum_{i=1}^{n} \gamma_i y_i \,. \tag{2}$$

It should be the case that the polytope includes all the codewords, and does not include any integral non-codewords. As such, a polytope $\mathcal{P}$ is called *proper* for code C if $\mathcal{P} \cap \{0,1\}^n = C$.

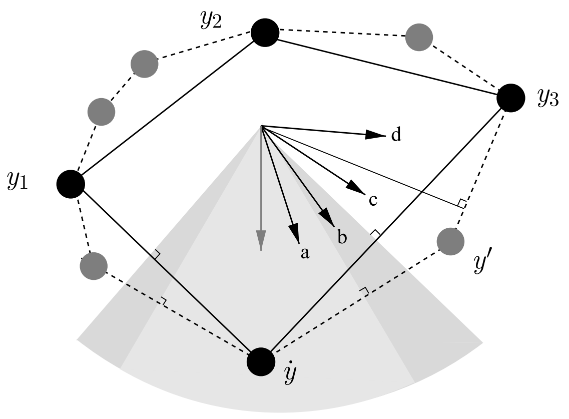
The LP decoder works as follows. Solve the LP in (2) to obtain $y_{\text{LP}} \in [0,1]^n$. If y_{LP} is integral (i. e., all elements are 0 or 1), then output y_{LP}. Otherwise, output "error". By the definition of a proper polytope, if the LP decoder outputs a codeword, it is guaranteed to be equal to the ML codeword y^*. This fact is known as the *ML certificate* property.

Comparing with ML Decoding

A successful decoder is one that outputs the original codeword transmitted over the channel, and so the quality of an algorithm is measured by the likelihood that this happens.

(Another common non-probabilistic measure is the *worst-case* performance guarantee, which measures how many bit-flips an algorithm can tolerate and still be guaranteed to succeed.) Note that y^* is the one *most likely* to be the transmitted codeword $\dot{y}$, but it is not always the case that $y^* = \dot{y}$. However, no decoder can perform better than an ML decoder, and so it is useful to use ML decoding as a basis for comparison.

Figure 1 provides a geometric perspective of LP decoding, and its relation to exact ML decoding. Both decoders use the same LP objective function, but over different constraint sets. In exact ML decoding, the constraint set is the convex hull C of codewords (i. e., the set of points that are convex combinations of codewords from C), whereas relaxed LP decoding uses the larger polytope $\mathcal{P}$. In Fig. 1, the four arrows labeled (a)–(d) correspond to different "noisy" versions of the LP objective function. (a) If there is very little noise, then the objective function points to the transmitted codeword $\dot{y}$, and thus both ML decoding and LP decoding succeed, since both have the transmitted codeword $\dot{y}$ as the optimal point. (b) If more noise is introduced, then ML decoding succeeds, but LP decoding fails, since the fractional vertex y' is optimal for the relaxation. (c) With still more noise, ML decoding fails, since y_3 is now optimal; LP decoding still has a fractional optimum y', so this error is in some sense "detected". (d) Finally, with a lot of noise, both ML decoding and LP decoding have y_3 as the optimum, and so both methods fail and the error is "undetected". Note that in the last two cases (c, d), when ML decoding fails, the failure of the LP decoder is in some sense the fault of the code itself, as opposed to the decoder.



LP Decoding, Figure 1
A decoding polytope $\mathcal{P}$ (*dotted line*) and the convex hull C (*solid line*) of the codewords $\dot{y}$, y_1, y_2, and y_3. Also shown are the four possible cases (a–d) for the objective function, and the normal cones to both $\mathcal{P}$ and C

Normal Cones and C-Symmetry

The (negative) *normal cones* at $\dot{y}$ (also called the *fundamental cone* [10]) is defined as follows:

$$N_{\dot{y}}(\mathcal{P}) = \left\{\gamma \in \mathbb{R}^n : \sum_i \gamma_i(y_i - \dot{y}_i) \geq 0 \text{ for all } y \in \mathcal{P}\right\},$$

$$N_{\dot{y}}(C) = \left\{\gamma \in \mathbb{R}^n : \sum_i \gamma_i(y_i - \dot{y}_i) \geq 0 \text{ for all } y \in C\right\} .$$

Note that $N_{\dot{y}}(\mathcal{P})$ corresponds to the set of cost vectors γ such that $\dot{y}$ is an optimal solution to (2). The set $N_{\dot{y}}(C)$ has a similar interpretation as the set of cost vectors γ for which $\dot{y}$ is the ML codeword. Since $\mathcal{P} \subset C$, it is immediate from the definition that $N_y(C) \supset N_y(\mathcal{P})$ for all $y \in C$. Fig. 1 shows these two cones and their relationship.

The success probability of an LP decoder is equal to the total probability mass of $N_{\dot{y}}(\mathcal{P})$, under the distribution on cost vectors defined by the channel. The success probability of ML decoding is similarly related to the probability mass in the normal cone $N_y(C)$. Thus, the discrepancy between the normal cones of $\mathcal{P}$ and C is a measure of the gap between exact ML and relaxed LP decoding.

This analysis is specific to a particular transmitted codeword $\dot{y}$, but one would like to apply it in general. When dealing with linear codes, for most decoders one can usually assume that an arbitrary codeword is transmitted, since the decision region for decoding success is symmetric. The same holds true for LP decoding (see [4] for proof), as long as the polytope $\mathcal{P}$ is *C-symmetric*, defined as follows:

Definition 1 A proper polytope $\mathcal{P}$ for the binary code C is C-**symmetric** if, for all $y \in \mathcal{P}$ and $\dot{y} \in C$, it holds that $y' \in \mathcal{P}$, where $y'_i = |y_i - \dot{y}_i|$.

Using a Dual Witness to Prove Error Bounds

In order to prove that LP decoding succeeds, one must show that $\dot{y}$ is the optimal solution to the LP in (2). If the code C is linear, and the relaxation is proper and C-symmetric, one can assume that $\dot{y} = 0^n$, and then show that 0^n is optimal. Consider the *dual* of the decoding LP in (2). If there is a feasible point of the dual LP that has the same cost (i. e., zero) as the point 0^n has in the primal, then 0^n must be an optimal point of the decoding LP. Therefore, to prove that the LP decoder succeeds, it suffices to exhibit a zero-cost point in the dual.[1]

[1]Actually, since the existence of the zero-cost dual point only proves that 0^n is one of possibly many primal optima, one needs to be a bit more careful, a minor issue deferred to more complete treatments of this material.

Key Results

LP decoders have mainly been studied in the context of Low-Density Parity-Check codes [9], and their generalization to expander codes [12]. LP decoders for Turbo codes [2] have also been defined, but the results are not as strong. This summary of key results gives bounds on the *word error rate (WER)*, which is the probability, over the noise in the channel, that the decoder does not output the transmitted word. These bounds are relative to specific *families* of codes, which are defined as infinite set of codes of increasing length whose rate is bounded from below by some constant. Here the bounds are given in asymptotic form (without constants instantiated), and only for the binary symmetric channel.

Many other important results that are not listed here are known for LP decoding and related notions. Some of these general areas are surveyed in the next section, but there is insufficient space to reference most of them individually; the reader is referred to [3] for a thorough bibliography.

Low-Density Parity-Check Codes

The polytope $\mathcal{P}$ for LDPC codes, first defined in [4,8,10], is based on the underlying *Tanner graph* of the code, and has a linear number of variables and constraints. If the Tanner graph expands sufficiently, it is known that LP decoding can correct a constant fraction of errors in the channel, and thus has an inverse exponential error rate. This was proved using a dual witness:

Theorem 1 (**[6]**) *For any rate $r > 0$, there is a constant $\epsilon > 0$ such that there exists a rate r family of low-density parity-check codes with length n where the LP decoder succeeds as long as at most ϵn bits are flipped by the channel. This implies that there exists a constant $\epsilon' > 0$ such that the word error rate under the BSC$_p$ with $p < \epsilon'$ is at most $2^{-\Omega(n)}$.*

Expander Codes

The *capacity* of a communication channel bounds from above the rate one can obtain from a family of codes and still get a word error rate that goes to zero as the code length increases. The notation C_p is used to denote the capacity of the BSC$_p$. Using a family of codes based on expanders [12], LP decoding can achieve rates that approach capacity. Compared to LDPC codes, however, this comes at the cost of increased decoding complexity, as the size of the LP is exponential in the gap between the rate and capacity.

Theorem 2 (**[7]**) *For any $p > 0$, and any rate $r < C_p$, there exists a rate r family of expander codes with length n such that the word error rate of LP decoding under the BSC$_p$ is at most $2^{-\Omega(n)}$.*

Turbo Codes

Turbo codes [2] have the advantage that they can be encoded in linear time, even in a streaming fashion. *Repeat-accumulate* codes are a simple form of Turbo code. The LP decoder for Turbo codes and their variants was first defined in [4,5], and is based on the *trellis* structure of the component *convolutional* codes. Due to certain properties of turbo codes it is impossible to prove bounds for turbo codes as strong as the ones for LDPC codes, but the following is known:

Theorem 3 (**[5]**) *There exists a rate $1/2 - o(1)$ family of repeat-accumulate codes with length n, and a constant $\epsilon > 0$, such that under the BSC$_p$ with $p < \epsilon$, the LP decoder has a word error rate of at most $n^{-\Omega(1)}$.*

Applications

The application of LP decoding that has received the most attention so far is for LDPC codes. The LP for this family of codes not only serves as an interesting alternative to more conventional iterative methods [15], but also gives a useful tool for analyzing those methods, an idea first explored in [8,10,14]. Iterative methods such as *belief propagation* use local computations on the Tanner graph to update approximations of the marginal probabilities of each code bit. In this type of analysis, the vertices of the polytope $\mathcal{P}$ are referred to as *pseudocodewords*, and tend to coincide with the fixed points of this iterative process. Other notions of pseudocodeword-like structures such as *stopping sets* are also known to coincide with these polytope vertices. Understanding these structures has also inspired the design of new codes for use with iterative and LP decoding. (See [3] for a more complete bibliography of this work).

The decoding method itself can be extended in many ways. By adding redundant information to the description of the code, one can derive tighter constraint sets to improve the error-correcting performance of the decoder, albeit at an increase in complexity. Adaptive algorithms that try to add constraints "on the fly" have also been explored, using branch-and-bound or other techniques. Also, LP decoding has inspired the use of other methods from optimization theory in decoding error-correcting codes. (Again, see [3] for references.)

Open Problems

The LP decoding method gives a simple, efficient and analytically tractable approach to decoding error-correcting codes. The results known to this point serve as a proof of concept that strong bounds are possible, but there are still important questions to answer. Although LP decoders can achieve capacity with decoding time polynomial in the length of the code, the complexity of the decoder still depends exponentially on the gap between the rate and capacity (as is the case for all other known provably efficient capacity-achieving decoders). Decreasing this dependence would be a major accomplishment, and perhaps LP decoding could help. Improving the fraction of errors correctable by LP decoding is also an important direction for further research.

Another interesting question is whether there exist constant-rate linear-distance code families for which one can formulate a polynomial-sized exact decoding LP. Put another way, is there a constant-rate linear-distance family of codes whose convex hulls have a polynomial number of facets? If so, then LP decoding would be equivalent to ML decoding for this family. If not, this is strong evidence that suboptimal decoding is inevitable when using good codes, which is a common belief.

An advantage to LP decoding is the *ML certificate* property mentioned earlier, which is not enjoyed by most other standard suboptimal decoders. This property opens up the possibility for a wide range of heuristics for improving decoding performance, some of which have been analyzed, but largely remain wide open.

LP decoding has (for the most part) only been explored for LDPC codes under memoryless symmetric channels. The LP for turbo codes has been defined, but the error bounds proved so far are not a satisfying explanation of the excellent performance observed in practice. Other codes and channels have gotten little, if any, attention.

Cross References

► Decoding Reed–Solomon Codes
► Learning Heavy Fourier Coefficients of Boolean Functions
► Linearity Testing/Testing Hadamard Codes
► List Decoding near Capacity: Folded RS Codes

Recommended Reading

1. Berlekamp, E., McEliece, R., van Tilborg, H.: On the inherent intractability of certain coding problems. IEEE Trans. Inf. Theory **24**, 384–386 (1978)
2. Berrou, C., Glavieux, A., Thitimajshima, P.: Near Shannon limit error-correcting coding and decoding: turbo-codes. In: Proc. IEEE Int. Conf. Comm. (ICC), pp. 1064–1070. Geneva, 23–26 May 1993
3. Boston, N., Ganesan, A., Koetter, R., Pazos, S., Vontobel, P.: Papers on pseudocodewords. HP Labs, Palo Alto. http://www.pseudocodewords.info.
4. Feldman, J.: Decoding Error-Correcting Codes via Linear Programming. Ph. D. thesis, Massachusetts Institute of Technology (2003)
5. Feldman, J., Karger, D.R.: Decoding turbo-like codes via linear programming. In: Proc. 43rd annual IEEE Symposium on Foundations of Computer Science (FOCS), Vancouver, 16–19 November 2002
6. Feldman, J., Malkin, T., Servedio, R.A., Stein, C., Wainwright, M.J.: LP decoding corrects a constant fraction of errors. In: Proc. IEEE International Symposium on Information Theory, Chicago, 27 June – 2 July 2004
7. Feldman, J., Stein, C.: LP decoding achieves capacity. In: Symposium on Discrete Algorithms (SODA '05), Vancouver, January (2005)
8. Feldman, J., Wainwright, M.J., Karger, D.R.: Using linear programming to decode linear codes. In: 37th annual Conf. on Information Sciences and Systems (CISS '03), Baltimore, 12–14 March 2003
9. Gallager, R.: Low-density parity-check codes. IRE Trans. Inform. Theory, IT-8 , pp. 21–28 (1962)
10. Koetter, R., Vontobel, P.: Graph covers and iterative decoding of finite-length codes. In: Proc. 3rd International Symposium on Turbo Codes and Related Topics, pp. 75–82, September 2003. Brest, France (2003)
11. MacWilliams, F.J., Sloane, N.J.A.: The Theory of Error Correcting Codes. North-Holland, Amsterdam (1981)
12. Sipser, M., Spielman, D.: Expander codes. IEEE Trans. Inf. Theory **42**, 1710–1722 (1996)
13. Vazirani, V.V.: Approximation Algorithms. Springer, Berlin (2003)
14. Wainwright, M., Jordan, M.: Variational inference in graphical models: the view from the marginal polytope. In: Proc. 41st Allerton Conf. on Communications, Control, and Computing, Monticello, October (2003)
15. Wiberg, N.: Codes and Decoding on General Graphs, Ph. D. thesis, Linkoping University, Sweden (1996)

M

Majority Equilibrium

2003; Chen, Deng, Fang, Tian

QIZHI FANG
Department of Mathematics, Ocean University of China, Qingdao, China

Keywords and Synonyms

Condorcet winner

Problem Definition

Majority rule is arguably the best decision mechanism for public decision making, which is employed not only in public management but also in business management. The concept of majority equilibrium captures such a democratic spirit in requiring that no other solutions would please more than half of the voters in comparison to it. The work of Chen, Deng, Fang, and Tian [1] considers a public facility location problem decided via a voting process under the majority rule on a discrete network. This work distinguishes itself from previous work by applying the computational complexity approach to the study of majority equilibrium. For the model with a single public facility located in trees, cycles, and cactus graphs, it is shown that the majority equilibrium can be found in linear time. On the other hand, when the number of public facilities is taken as the input size (not a constant), finding a majority equilibrium is shown to be $\mathcal{NP}$-hard.

Consider a network $G = ((V, \omega), (E, l))$ with vertex and edge weight functions $\omega : V \to R^+$ and $l : E \to R^+$, respectively. Each vertex $i \in V$ represents a community, and $\omega(i)$ represents the number of voters that reside there. For each $e \in E$, $l(e) > 0$ represents the length of the road $e = (i, j)$ connecting two communities i and j. For two vertices $i, j \in V$, the distance between i and j, denoted by $d_G(i, j)$, is the length of a shortest path joining them. The location of a public facility such as a library, community center, etc., is to be determined by the public via a voting process under the majority rule. Here, each member of the community desires to have the public facility close to himself, and the decision has to be agreed upon by a majority of the voters. Denote the vertex set of G by $V = \{v_1, v_2, \cdots, v_n\}$. Then each $v_i \in V$ has a preference order $\geq_i$ on V induced by the distance on G. That is, $x \geq_i y$ if and only if $d_G(v_i, x) \leq d_G(v_i, y)$ for two vertices $x, y \in V$; similarly, $x >_i y$ if and only if $d_G(v_i, x) < d_G(v_i, y)$. Under such a preference profile, four types of majority equilibrium, called Condorcet winners, are defined as follows.

Definition 1 Let $v_0 \in V$, then v_0 is called:

(1) *a weak quasi-Condorcet winner*, if for every $u \in V$ distinct of v_0,

$$\omega(\{v_i \in V : v_0 \geq_i u\}) \geq \sum_{v_i \in V} \omega(v_i)/2;$$

(2) *a strong quasi-Condorcet winner*, if for every $u \in V$ distinct of v_0,

$$\omega(\{v_i \in V : v_0 \geq_i u\}) > \sum_{v_i \in V} \omega(v_i)/2;$$

(3) *a weak Condorcet winner*, if for every $u \in V$ distinct of v_0,

$$\omega(\{v_i \in V : v_0 >_i u\}) \geq \omega(\{v_i \in V : u >_i v_0\});$$

(4) *a strong Condorcet winner*, if for every $u \in V$ distinct of v_0,

$$\omega(\{v_i \in V : v_0 >_i u\}) > \omega(\{v_i \in V : u >_i v_0\}).$$

Under the majority voting mechanism described above, the problem is to develop efficient ways for determining the existence of Condorcet winners and finding such a winner when one exists.

Problem 1 ***(Finding Condorcet Winners)***
INPUT: A network $G = ((V, w), (E, l))$.
OUTPUT: A Condorcet winner $v \in V$, or nonexistence of Condorcet winners.

Key Results

The mathematical results given in this section depend deeply on the understanding of combinatorial structures of underlying networks. Theorem 1, 2, and 3 below are given for weak quasi-Condorcet winners in the model with a single facility to be located. Other kinds of Condorcet winners can be discussed similarly.

Theorem 1 *Every tree has one weak quasi-Condorcet winner, or two adjacent weak quasi-Condorcet winners, which can be found in linear time.*

Theorem 2 *Let C_n be a cycle of order n with vertex-weight function $\omega : V(C_n) \rightarrow R^+$. Then $v \in V(C_n)$ is a weak quasi-Condorcet winner of C_n if and only if the weight of each $\lfloor \frac{n+1}{2} \rfloor$-interval containing v is at least $\frac{1}{2} \sum_{v \in C_n} \omega(v)$. Furthermore, the problem of finding a weak quasi-Condorcet winner of C_n is solvable in linear time.*

Given a graph $G = (V, E)$, a vertex v of G is a *cut vertex* if $E(G)$ can be partitioned into two nonempty subsets E_1 and E_2 such that the induced graphs $G[E_1]$ and $G[E_2]$ have just the vertex v in common. A *block* of G is a connected subgraph of G that has no cut vertices and is maximal with respect to this property. Every graph is the union of its blocks. A graph G is called a *cactus graph*, if G is a connected graph in which each block is an edge or a cycle.

Theorem 3 *The problem of finding a weak quasi-Condorcet winner of a cactus graph with vertex-weight function is solvable in linear time.*

In general, the problem can be extended to the cases where a number of public facilities are required to be located during one voting process, and the definitions of Condorcet winners can also be extended accordingly. In such cases, the public facilities may be of the same type, or different types; and the utility functions of the voters may be of different forms.

Theorem 4 *If there are a bounded constant number of public facilities to be located at one voting process under the majority rule, then the problem of finding a Condorcet winner (any of the four types) can be solved in polynomial time.*

Theorem 5 *If the number of public facilities to be located is not a constant but considered as the input size, the problem of finding a Condorcet winner is $\mathcal{NP}$-hard; and the corresponding decision problem: deciding whether a candidate set of public facilities is a Condorcet winner, is co-$\mathcal{NP}$-complete.*

Applications

Damange [2] first reviewed continuous and discrete spatial models of collective choice, aiming at characterizing the public facility location problem as a result of the pubic voting process. Although the network models in Chen et al. [1] have been studied for some problems in economics [3,4], the principal point of departure in Chen et al.'s work is the computational complexity and algorithmic approach. This approach can be applied to more general public decision-making processes.

For example, consider a public road repair problem, pioneered by Tullock [5] to study redistribution of tax revenue under a majority rule system. An edge-weighted graph $G = (V, E, w)$ represents a network of local roads, where the weight of each edge represents the cost of repairing the road. There is also a distinguished vertex $s \in V$ representing the entry point to the highway system. The majority decision problem involves a set of agents $A \subseteq V$ situated at vertices of the network who would choose a subset F of edges. The cost of repairing F, which is the sum of the weights of edges in F, will be shared by all n agents, each an n-th of the total. In this model, a majority stable solution under the majority rule is a subset $F \subseteq E$ that connects s to a subset $A_1 \subset A$ of agents with $|A_1| > |A|/2$ such that no other solution H connecting s to a subset of agents $A_2 \subset A$ with $|A_2| > |A|/2$ satisfies the conditions that $\sum_{e \in H} w(e) \leq \sum_{e \in F} w(e)$, and for each agent in A_2, its shortest path to s in solution H is not longer than that in solution F, and at least one of the inequalities is strict. It is shown in Chen et al. [1] that for this model, finding a majority equilibrium is $\mathcal{NP}$-hard for general networks, and is polynomially solvable for tree networks.

Cross References

- ► General Equilibrium
- ► Leontief Economy Equilibrium
- ► Local Search for K-medians and Facility Location

Recommended Reading

1. Chen, L., Deng, X., Fang, Q., Tian, F.: Majority equilibrium for public facility allocation. Lect. Notes Comput. Sci. **2697**, 435–444 (2002)
2. Demange, G.: Spatial Models of Collective Choice. In: Thisse, J.F., Zoller, H.G. (eds.) Locational Analysis of Public Facilities, North-Holland Publishing Company, North Holland, Amsterdam (1983)
3. Hansen, P., Thisse, J.F.: Outcomes of voting and planning: condorcet, weber and rawls locations. J. Publ. Econ. **16**, 1–15 (1981)
4. Schummer, J., Vohra, R.V.: Strategy-proof location on a network. J. Econ. Theor. **104**, 405–428 (2002)
5. Tullock, G.: Some problems of majority voting. J. Polit. Econ. **67**, 571–579 (1959)

Market Games and Content Distribution

2005; Mirrokni

VAHAB S. MIRROKNI
Theory Group, Microsoft Research, Redmond, WA, USA

Keywords and Synonyms

Market sharing games; Valid-Utility games; Congestion games; Stable matching

Problem Definition

This chapter studies market games for their performance and convergence of the equilibrium points. The main application is the content distribution in cellular networks in which a service provider needs to provide data to users. The service provider can use several cache locations to store and provide the data. The assumption is that cache locations are selfish agents (resident subscribers) who want to maximize their own profit. Most of the results apply to a general framework of monotone two-sided markets.

Uncoordinated Two-Sided Markets

Various economic interactions can be modeled as two-sided markets. A two-sided market consists of two disjoint groups of agents: active agents and passive agents. Each agent has a preference list over the agents of the other side, and can be matched to one (or many) of the agents in the other side. A central solution concept to these markets are *stable matchings*, introduced by Gale and Shapley [5]. It is well known that stable matchings can be achieved using a centralized polynomial-time algorithm. Many markets, however, do not have any centralized matching mechanism to match agents. In those markets, matchings are formed by actions of self-interested agents. Knuth [9] introduced uncoordinated two-sided markets. In these markets, cycles of better or best responses exist, but random better response and best response dynamics converge to a stable matching with probability one [2,10,14]. Our model for content distribution corresponds to a special class of uncoordinated two-sided markets that is called *the distributed caching games*.

Before introducing the distributed caching game as an uncoordinated two-sided market, the distributed caching problem and some game theoretic notations are defined.

Distributed Caching Problem

Let U be a set of n cache locations with given available capacities A_i and given available bandwidths B_i for each cache location i. There are k request types;[1] each request type t has a size a_t $(1 \le t \le k)$. Let H be a set of m requests with a reward R_j, a required bandwidth b_j, a request type t_j for each request j, and a cost c_{ij} for connecting each cache location i to each request j. The profit of providing request j by cache location i is $f_{ij} = R_j - c_{ij}$. A cache location i can service a set of requests S_i, if it satisfies the *bandwidth constraint*: $\sum_{j \in S_i} b_j \le B_i$, and the *capacity constraint*: $\sum_{t \in \{t_j | j \in S_i\}} a_t \le A_i$ (this means that the sum of the sizes of the request types of the requests in cache location i should be less than or equal to the available capacity of cache location i). A set S_i of requests is feasible for cache location i if it satisfies both of these constraints. The goal of the DCP problem is to find a feasible assignment of requests to cache locations to maximize the total profit; i. e., the total reward of requests that are provided minus the connection costs of these requests.

Strategic Games

A *strategic game* $\mathcal{G}$ is defined as a tuple $\mathcal{G}(U, \{F_i | i \in U\}, \{\alpha_i() | i \in U\})$ where (i) U is the set of n players or agents, (ii) F_i is a family of feasible *(pure) strategies* or *actions* for player i and (iii) $\alpha_i : \Pi_{i \in U} F_i \to \mathbb{R}^+ \cup \{0\}$ is the (private) *payoff* or *utility* function for agent i, given the set of strategies of all players. Player i's strategy is denoted by $s_i \in F_i$. A *strategy profile* or a (strategy) *state*, denoted by $S = (s_1, s_2, \dots, s_n)$, is a vector of strategies of players. Also let $S \oplus s'_i := (s_1, \dots, s_{i-1}, s'_i, s_{i+1}, \dots, s_k)$.

Best-Response Moves

In a *non-cooperative* game, each agent wishes to maximize its own payoff. For a strategy profile $S = (s_1, s_2, \dots, s_n)$, a *better response move* of player i is a strategy s'_i such that $\alpha_i(S \oplus s'_i) \ge \alpha_i(S)$. In a *strict better response move*, the above inequality is strict. Also, for a strategy profile $S = (s_1, s_2, \dots, s_n)$ a *best response* of player i in S is a better response move $s^*_i \in F_i$ such that for any strategy $s_i \in F_i$, $\alpha_i(S \oplus s^*_i) \ge \alpha_i(S \oplus s_i)$.

Nash Equilibria

A pure strategy Nash equilibrium (PSNE) of a strategic game is a strategy profile in which each player plays his best response.

[1] Request type can be thought of as different files that should be delivered to clients.

State Graph

The state graph, $\mathcal{D} = (\mathcal{F}, \mathcal{E})$, of a strategic game $\mathcal{G}$, is an arc-labeled directed graph, where the vertex set $\mathcal{F}$ corresponds to the set of strategy profiles or states in $\mathcal{G}$, and there is an arc from state S to state S' with label i if the only difference between S and S' is in the strategy of player i; and player i plays one of his best responses in strategy profile S'. A *best-response walk* is a directed walk in the state graph.

Price of Anarchy

Given a strategic game, $\mathcal{G}(U, \{F_i | i \in U\}, \{\alpha() | i \in U\})$, and a maximization social function $\gamma : \Pi_{i \in U} F_i \to \mathbb{R}$, the price of anarchy, denoted by $\mathsf{poa}(\mathcal{G}, \gamma)$, is the worst ratio between the social value of a pure Nash equilibrium and the optimum.

Distributed Caching Games

The distributed caching game can be formalized as a two-sided market game: active agents correspond to n resident subscribers or cache locations, and passive agents correspond to m requests from transit subscribers. Formally, given an instance of the DCP problem, a strategic game $\mathcal{G}(U, \{F_i | i \in U\}, \{\alpha_i | i \in U\})$ is defined as follows. The set of players (or active agents) U is the set of cache locations. The family of feasible strategies F_i of a cache location i is the family of subsets s_i of requests such that $\sum_{j \in s_i} b_j \le B_i$ and $\sum_{t \in \{t_j | j \in s_i\}} a_t \le A_i$. Given a vector $S = (s_1, s_2, \dots, s_n)$ of strategies of cache locations, the *favorite* cache locations for request j, denoted by $\mathsf{FAV}(j)$, is the set of cache locations i such that $j \in s_i$ and f_{ij} has the maximum profit among the cache locations that have request j in their strategy set, i. e., $f_{ij} \ge f_{i'j}$ for any i' such that $j \in s_{i'}$. For a strategy profile $S = (s_1, \dots, s_n)$ $\alpha_i(S) = \sum_{j : i \in \mathsf{FAV}(j)} f_{ij} / |\mathsf{FAV}(j)|$. Intuitively, the above definition implies that the profit of each request goes to the cache locations with the minimum connection cost (or equivalently with the maximum profit) among the set of cache locations that provide this request. If more than one cache location have the maximum profit (or minimum connection cost) for a request j, the profit of this request is divided equally between these cache locations. The payoff of a cache location is the sum of profits from the requests it actually serves. A player i *serves* a request j if $i \in \mathsf{FAV}(j)$. The social value of strategy profile S, denoted by $\gamma(S)$, is the sum of profits of all players. This value $\gamma(S)$ is a measure of the efficiency of the assignment of requests and request types to cache locations.

Special Cases

In this paper, the following variants and special cases of the DCP problem are also studied: The CapDCP problem is a special case of DCP problem without bandwidth constraints. The BanDCP problem is a special case of DCP problem without capacity constraints. In the uniform BanDCP problem, the bandwidth consumption of all requests is the same. In the uniform CapDC problem, the size of all request types is the same.

Many-to-One Two-Sided Markets with Ties

In the distributed caching game, active and passive agents correspond to cache locations and requests respectively. The set of feasible strategies for each active agent correspond to a set of solutions to a packing problem. Moreover, the preferences of both active and passive agents is determined from the profit of requests to cache locations. In many-to-one two-sided markets, the preference of passive and active agents as well as the feasible family of strategies are arbitrary. The preference list of agents may have ties as well.

Monotone and Matroid Markets

In *monotone many-to-one two-sided markets*, the preferences of both active and passive agents are determined based on payoffs $p_{ij} = p_{ji}$ for each active agent i and passive agent j (similar to the DCP game). An agent i prefers j to j' if $p_{ij} > p_{ij'}$. In *matroid two-sided markets*, the feasible set of strategies of each active agent is the set of independent sets of a matroid. Therefore, uniform BanDCP game is a matroid two-sided market game.

Key Results

In this section, the known results for these problems are summarized.

Centralized Approximation Algorithm

The distributed caching problem generalizes the multiple knapsack problem and the generalized assignment problem [3] and as a result is an APX-hard problem.

Theorem 1 ([4]) *There exists a linear programming based $1 - \frac{1}{e}$-approximation algorithm and a local search $\frac{1}{2}$-approximation algorithm for the* DCP *problem.*

The $1 - \frac{1}{e}$-approximation for this problem is based on rounding an exponentially large configuration linear program [4]. On the basis of some reasonable complexity theoretic assumptions, this approximation factor of $1 - \frac{1}{e}$ is tight for this problem. More formally,

Theorem 2 ([4]) *For any $\epsilon > 0$, there exists no $1 - \frac{1}{e} - \epsilon$-approximation algorithm for the* DCP *problem unless NP $\subseteq DTIME(n^{O(\log\log n)})$.*

Price of Anarchy

Since the DCP game is a strategic game, it possesses mixed Nash equilibria [12]. The DCP game is a valid-utility game with a submodular social function as defined by Vetta [16]. This implies that the performance of any mixed Nash equilibrium of this game is at least $\frac{1}{2}$ of the optimal solution.

Theorem 3 ([4,11]) *The* DCP *game is a valid-utility game and the price of anarchy for mixed Nash equilibria is $\frac{1}{2}$. Moreover, this result holds for all monotone many-to-one two-sided markets with ties.*

A direct proof of the above price of anarchy bound for the DCP game can be found in [11].

Pure Nash Equilibria: Existence and Convergence

This part surveys known results for existence and convergence of pure Nash equilibria.

Theorem 4 ([11]) *There are instances of the* IBDC *game that have no pure Nash equilibrium.*

Since, IBDC is a special case of CapDCP, the above theorem implies that there are instances of the CapDCP game that have no pure Nash equilibrium. In the above theorem, the bandwidth consumption of requests are not uniform, and this was essential in finding the example. The following gives theorems for the uniform variant of these games.

Theorem 5 ([1,11]) *Any instance of the uniform* BanDCP *game does not contain any cycle of strict best-response moves, and thus possess a pure Nash equilibrium. On the other hand, there are instances of the uniform* CapDCP *game with no pure Nash equilibria.*

The above result for the uniform BanDCP game can be generalized to matroid two-sided markets with ties as follows.

Theorem 6 ([1]) *Any instance of the monotone matroid two-sided market game with ties is a potential game, and possess pure Nash equilibria. Moreover, any instance of the matroid two-sided market game with ties possess pure Nash equilibria.*

Convergence Time to Equilibria

This section proves that there are instances of the uniform CapDCP game in which finding a pure Nash equilibrium is PLS-hard [8]. The definition of PLS-hard problems can be found in papers by Yannakakis et al. [8,15].

Theorem 7 ([11]) *There are instances of the uniform* CapDCP *game with pure Nash equilibria*[2] *for which finding a pure Nash equilibrium is PLS-hard.*

Using the above proof and a result of Schaffer and Yannakakis [13,15], it is possible to show that in some instances of the uniform CapDCP game, there are states from which all paths of best responses have exponential length.

Corollary 1 ([11]) *There are instances of the uniform* CapDCP *game that have pure Nash equilibria with states from which any sequence of best-response moves to any pure Nash equilibrium (or sink equilibrium) has an exponential length.*

The above theorems show exponential convergence to pure Nash equilibria in general DCP games. For the special case of the uniform BanDCP game, the following is a positive result for the convergence time to equilibria.

Theorem 8 ([2]) *The expected convergence time of a random best-response walk to pure Nash equilibria in matroid monotone two-sided markets (without ties) is polynomial.*

Since the uniform BanDCP game is a special case of matroid monotone two-sided markets with ties, the above theorem indicates that for the BanDCP game with no tie in the profit of requests, the convergence time of a random best-response walk is polynomial. Finally, we state a theorem about the convergence time of the general (non-monotone) matroid two-sided market games.

Theorem 9 ([2]) *In the matroid two-sided markets (without ties), a random best response dynamic of players may cycle, but it converges to a Nash equilibrium with probability one. However, it may take exponential time to converge to a pure Nash equilibrium.*

Pure Nash equilibria of two-sided market games correspond to stable matchings in two-sided markets and vice-versa [2]. The fact that better response dynamics of players in two-sided market games may cycle, but will converge to a stable matching has been proved in [9,14]. Ackermann et al. [2] extend these results for best-response dynamics, and show an exponential lower bound for expected convergence time to pure Nash equilibria.

[2]It is also possible to say that finding a *sink equilibrium* is PLS-hard. A sink equilibrium is a set of strategy profiles that is closed under best-response moves. A pure equilibrium is a sink equilibrium with exactly one profile. This equilibrium concept is formally defined in [7].

Applications

The growth of the Internet, the World Wide Web, and wide-area wireless networks allow an increasing number of users to access vast amounts of information in different geographic areas. As one of the most important functions of the service provider, content delivery can be performed by caching popular items in cache locations close to the users. Performing such a task in a decentralized manner in the presence of self-interested entities in the system can be modeled as an uncoordinated two-sided market game.

The 3G subscriber market can be categorized into groups with shared interest in location-based services, e. g., the preview of movies in a theater or scenes of the mountain nearby. Since the 3G radio resources are limited, it is expensive to repeatedly transmit large quantities of data over the air interface from the base station (BS). It is more economical for the service provider to offload such repeated requests on to the ad-hoc network comprised of its subscribers where some of them recently acquired a copy of the data. In this scenario, the goal for the service provider is to give incentives for peer subscribers in the system to cache and forward the data to the requesting subscribers. Since each data item is large in size and transit subscribers are mobile, we assume that the data transfer occurs in a close range of a few hops.

In this setting, envision a system consisting of two groups of subscribers: resident and transit subscribers. Resident subscribers are less mobile and mostly confined to a certain geographical area. Resident subscribers have incentives to cache data items that are specific to this geographical region since the service provider gives monetary rewards for satisfying the queries of transit subscribers. Transit subscribers request their favorite data items when they visit a particular region. Since the service provider does not have knowledge of the spatial and temporal distribution of requests, it is difficult if not impossible for the provider to stipulate which subscriber should cache which set of data items. Therefore, the decision of what to cache is left to each individual subscriber. The realization of this content distribution system depends on two main issues. First, since subscribers are selfish agents, they may act to increase their individual payoff and decrease the performance of the system. Here, we provide a framework for which we can prove that in an equilibrium situation of this framework, we use the performance of the system efficiently. The second issue is that the payoff of each request for each agent must be a function of the set of agents that have this request in their strategy, since these agents compete on this request and the profit of this request should be divided among these agents in an appropriate way. Therefore, each selfish agent may change the set of items it cached in response to the set of items cached by others. This model leads to a non-cooperative caching scenario that can be modeled on a two-sided market game, studied and motivated in the context of market sharing games and distributed caching games [4,6,11].

Open Problems

It is known that there exist instances of the distributed caching game with no pure Nash equilibria. It is also known that best response dynamics of players may take exponential time to converge to pure Nash equilibria. An interesting question is to study the performance of sink equilibria [7,11] or the price of sinking [7,11] for these games. The distributed caching game is a valid-utility game. Goemans, Mirrokni, and Vetta [7] show that despite the price of anarchy of $\frac{1}{2}$ for valid-utility games, the performance of sink equilibria (or price of sinking) for these games is $\frac{1}{n}$. We conjecture that the price of sinking for DCP games is a constant. Moreover, it is interesting to show that after polynomial rounds of best responses of players the approximation factor of the solution is a constant. We know that one round of best responses of players is not sufficient to get constant-factor solutions. It might be easier to show that after a polynomial number of random best responses of players, the expected total profit of players is at least a constant factor of the optimal solution. Similar positive results for sink equilibria and random best responses of players are known for congestion games [7,11].

The complexity of verifying if a given state of the distributed caching game is in a sink equilibrium or not is an interesting question to explore. Also, given a distributed caching game (or a many-to-one two-sided market game), an interesting problem is to check if the set of all sink equilibria is pure Nash equilibria or not. Finally, an interesting direction of research is to classify classes of two-sided market games for which pure Nash equilibria exists or best-response dynamics of players converge to a pure Nash equilibrium.

Cross References

▶ Stable Marriage
▶ Stable Marriage with Ties and Incomplete Lists
▶ Best Response Algorithms for Selfish Routing

Recommended Reading

1. Ackermann, H., Goldberg, P., Mirrokni, V., Röglin, H., Vöcking, B.: A unified Approach to Congestion Games and Two-sided markets. In: 3rd Workshop of Internet Economics (WINE), pp. 30–41. San Diego, CA, USA (2007)

2. Ackermann, H., Goldberg, P., Mirrokni, V., Röglin, H., Vöcking, B.: Uncoordinated two-sided markets. ACM Electronic Commerce (ACM EC) (2008)
3. Chekuri, C., Khanna, S.: A PTAS for the multiple knapsack problem. In 11th ACM-SIAM Symp. on Discrete Algorithms (SODA), pp. 213–222 (2000)
4. Fleischer, L., Goemans, M., Mirrokni, V.S., Sviridenko, M.: Tight approximation algorithms for maximum general assignment problems. In: Proceedings of the 16th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA), pp. 611–620 (2006)
5. Gale, D., Shapley, L.: College admissions and the stability of marriage. Am. Math. Mon. **69**, 9–15 (1962)
6. Goemans, M., Li, L., Mirrokni, V.S., Thottan, M.: Market sharing games applied to content distribution in ad-hoc networks. In: Proceedings of the 5th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc), pp. 1020–1033 (2004)
7. Goemans, M., Mirrokni, V.S., Vetta, A.: Sink equilibria and convergence. In: 46th Conference on Foundations of Computer Science (FOCS), pp. 123–131 (2005)
8. Johnson, D., Papadimitriou, C.H., Yannakakis, M.: How easy is local search? J. Comp. Syst. Sci. **37**, 79–100 (1988)
9. Knuth, D.: Marriage Stables et leurs relations avec d'autres problèmes Combinatories. Les Presses de l'Université de Montréal (1976)
10. Kojima, F., Unver, Ü.: Random paths to pairwise stability in many-to-many matching problems: A study on market equilibration. Intern. J. Game Theor. (2006)
11. Mirrokni, V.S.: Approximation Algorithms for Distributed and Selfish Agents. Ph.D. thesis, Massachusetts Institute of Technology (2005)
12. Nash, J.F.: Non-cooperative games. Ann. Math. **54**, 268–295 (1951)
13. Papadimitriou, C.H., Schaffer, A., Yannakakis, M.: On the complexity of local search. In: 22nd Symp. on Theory of Computing (STOC), pp. 438 – 445 (1990)
14. Roth, A.E., Vande Vate, J.H.: Random paths to stability in two-sided matching. Econometrica **58**(6), 1475–1480 (1990)
15. Schaffer, A. Yannakakis, M.: Simple local search problems that are hard to solve. SIAM J. Comput. **20**(1), 56–87 (1991)
16. Vetta, A.: Nash equilibria in competitive societies, with applications to facility location, traffic routing and auctions. In: 43rd Symp. on Foundations of Computer Science (FOCS), pp. 416–425 (2002)

Max Cut

1994; Goemans, Williamson
1995; Goemans, Williamson

ALANTHA NEWMAN
Department of Algorithms and Complexity,
Max-Planck Institute for Computer Science,
Saarbrücken, Germany

Keywords and Synonyms

Maximum bipartite subgraph

Problem Definition

Given an undirected edge-weighted graph, $G = (V, E)$, the maximum cut problem (MAX-CUT) is to find a bipartition of the vertices that maximizes the weight of the edges crossing the partition. If the edge weights are non-negative, then this problem is equivalent to finding a maximum weight subset of the edges that forms a bipartite subgraph, i. e. the maximum bipartite subgraph problem. All results discussed in this article assume non-negative edge weights. MAX-CUT is one of Karp's original NP-complete problems [19]. In fact, it is NP-hard to approximate to within a factor better than 16/17[16,33].

For nearly twenty years, the best-known approximation factor for MAX-CUT was half, which can be achieved by a very simple algorithm: Form a set S by placing each vertex in S with probability half. Since each edge crosses the cut $(S, V \setminus S)$ with probability half, the expected value of this cut is half the total edge weight. This implies that for any graph, there exists a cut with value at least half of the total edge weight. In 1976, Sahni and Gonzalez presented a deterministic half-approximation algorithm for MAX-CUT, which is essentially a de-randomization of the aforementioned randomized algorithm [31]: Iterate through the vertices and form sets S and $\bar{S}$ by placing each vertex in the set that maximizes the weight of cut $(S, \bar{S})$ thus far. After each iteration of this process, the weight of this cut will be at least half of the weight of the edges with both endpoints in $S \cup \bar{S}$.

This simple half-approximation algorithm uses the fact that for any graph with non-negative edge weights, the total edge weight of a given graph is an upper bound on the value of its maximum cut. There exist classes of graphs for which a maximum cut is arbitrarily close to half the total edge weight, i. e. graphs for which this "trivial" upper bound can be close to twice the true value of an optimal solution. An example of such a class of graphs are complete graphs on n vertices, K_n. In order to obtain an approximation factor better than half, one must be able to compute an upper bound on the value of a maximum cut that is better, i. e. smaller, than the trivial upper bound for such classes of graphs.

Linear Programming Relaxations

For many optimization (maximization) problems, linear programming has been shown to yield better (upper) bounds on the value of an optimal solution than can be obtained via combinatorial methods. There are several well-studied linear programming relaxations for MAX-CUT. For example, a classical integer program has a variable x_e for each edge and a constraint for each odd cycle, requiring that an odd cycle C contribute at most $|C| - 1$ edges to an optimal solution.

$$\begin{aligned}&\max \sum_{e\in E} w_e x_e\\&\sum_{e\in C} x_e \le |C| - 1 \quad \forall \text{ odd cycles } C\\&x_e \in \{0,1\}\,.\end{aligned}$$

The last constraint can be relaxed so that each x_e is required to lie between 0 and 1, but need not be integral, i. e. $0 \le x_e \le 1$. Although this relaxation may have exponentially many constraints, there is a polynomial-time separation oracle (equivalent to finding a minimum weight odd-cycle), and thus, the relaxation can be solved in polynomial time [13]. Another classical integer program contains a variable x_{ij} for each pair of vertices. In any partition of the vertices, either zero or two edges from a 3-cycle cross the cut. This requirement is enforced in the following integer program. If edge $(i, j) \notin E$, then w_{ij} is set to 0.

$$\begin{aligned}&\max \sum_{i,j\in V} w_{ij}x_{ij}\\&x_{ij} + x_{jk} + x_{ki} \le 2 \quad \forall i,j,k \in V\\&x_{ij} + x_{jk} - x_{ki} \ge 0 \quad \forall i,j,k \in V\\&\qquad\qquad x_{ij} \in \{0,1\}\,.\end{aligned}$$

Again, the last constraint can be relaxed so that each x_{ij} is required to lie between 0 and 1. In contrast to the aforementioned cycle-constraint based linear program, this linear programming relaxation has a polynomial number of constraints.

Both of these relaxations actually have the same optimal value for any graph with non-negative edge weights [3,26,30]. (For a simplified proof of this, see [25].) Poljak showed that the integrality gap for each of these relaxations is arbitrarily close to 2 [26]. In other words, there are classes of graphs that have a maximum cut containing close to half of the edges, but for which each of the above relaxations yields an upper bound close to all the edges, i. e. no better than the trivial "all-edges" bound. In particular, graphs with a maximum cut close to half the edges and with high girth can be used to demonstrate this gap. A comprehensive look at these linear programming relaxations is contained in the survey of Poljak and Tuza [30].

Eigenvalue Upper Bounds

Delorme and Poljak [7] presented an eigenvalue upper bound on the value of a maximum cut, which was a strengthened version of a previous eigenvalue bound considered by Mohar and Poljak [24]. Computing Delorme and Poljak's upper bound is equivalent to solving an eigenvalue minimization problem. They showed that their bound is computable in polynomial time with arbitrary precision. In a series of work, Delorme, Poljak and Rendl showed that this upper bound behaves "differently" from the linear-programming-based upper bounds. For example, they studied classes of sparse random graphs (e. g. $G(n, p)$ with $p = 50/n$) and showed that their upper bound is close to optimal on these graphs [8]. Since graphs of this type can also be used to demonstrate an integrality gap arbitrarily close to 2 for the aforementioned linear programming relaxations, their work highlighted contrasting behavior between these two upper bounds. Further computational experiments on other classes of graphs gave more evidence that the bound was indeed stronger than previously studied bounds [27,29]. Delorme and Poljak conjectured that the 5-cycle demonstrated the worst-case behavior for their bound: a ratio of $32/(25 + 5\sqrt{5}) \approx .88445$ between their bound and the optimal integral solution. However, they could not prove that their bound was strictly less than twice the value of a maximum cut in the worst case.

Key Results

In 1994, Goemans and Williamson presented a randomized .87856-approximation algorithm for MAX-CUT [11]. Their breakthrough work was based on rounding a semidefinite programming relaxation and was the first use of semidefinite programming in approximation algorithms. Poljak and Rendl showed that the upper bound provided by this semidefinite relaxation is equivalent to the eigenvalue bound of Delorme and Poljak [28]. Thus, Goemans and Williamson's proved that the eigenvalue bound of Delorme and Poljak is no more than 1.138 times the value of a maximum cut.

A Semidefinite Relaxation

MAX-CUT can be formulated as the following quadratic integer program, which is NP-hard to solve. Each vertex $i \in V$ is represented by a variable y_i, which is assigned either 1 or -1 depending on which side of the cut it occupies.

$$\begin{aligned}&\max \frac{1}{2} \sum_{(i,j)\in E} w_{ij}(1 - y_i y_j)\\&y_i \in \{-1, 1\} \quad \forall i \in V\,.\end{aligned}$$

Goemans and Williamson considered the following relaxation of this integer program, in which each vertex is represented by a unit vector.

$$\max \frac{1}{2} \sum_{(i,j)\in E} w_{ij}(1 - v_i \cdot v_j)$$
$$v_i \cdot v_i = 1 \quad \forall i \in V$$
$$v_i \in \mathcal{R}^n \quad \forall i \in V \,.$$

They showed that this relaxation is equivalent to a semidefinite program. Specifically, consider the following semidefinite relaxation:

$$\max \frac{1}{2} \sum_{(i,j)\in E} w_{ij}(1 - y_{ij})$$
$$y_{ii} = 1 \quad \forall i \in V$$
$$Y \quad \text{positive semidefinite} \,.$$

The equivalence of these two relaxations is due to the fact that a matrix Y is positive semidefinite if and only if there is a matrix B such that $B^{\mathrm{T}}B = Y$. The latter relaxation can be solved to within arbitrary precision in polynomial time via the Ellipsoid Algorithm, since it has a polynomial-time separation oracle [14]. Thus, a solution to the first relaxation can be obtained by finding a solution to the second relaxation and finding a matrix B such that $B^{\mathrm{T}}B = Y$. If the columns of B correspond to the vectors $\{v_i\}$, then $y_{ij} = v_i \cdot v_j$, yielding a solution to the first relaxation.

Random-Hyperplane Rounding

Goemans and Williamson showed how to round the semidefinite programming relaxation of MAX-CUT using a new technique that has since become known as "random-hyperplane rounding" [11]. First obtain a solution to the first relaxation, which consists of a set of unit vectors $\{v_i\}$, one vector for each vertex. Then choose a random vector $r \in \mathcal{R}^n$ in which each coordinate of r is chosen from the standard normal distribution. Finally, set $S = \{i | v_i \cdot r \geq 0\}$ and output the cut $(S, V \setminus S)$.

The probability that a particular edge $(i, j) \in E$ crosses the cut is equal to the probability that the dot products $v_i \cdot r$ and $v_j \cdot r$ differ in sign. This probability is exactly equal to θ_{ij}/π, where θ_{ij} is the angle between vectors v_i and v_j. Thus, the expected weight of edges crossing the cut is equal to $\sum_{(i,j)\in E} \theta_{ij}/\pi$. How large is this compared to the objective value given by the semidefinite programming relaxation, i. e. what is the approximation ratio?

Define α_{gw} as the worst-case ratio of the expected contribution of an edge to the cut, to its contribution to the objective function of the semidefinite programming relaxation. In other words: $\alpha_{gw} = \min_{0\leq\theta\leq\pi} \frac{2}{\pi} \frac{\theta}{1-\cos\theta}$. It can be shown that $\alpha_{gw} > .87856$. Thus, the expected value of a cut is at least $\alpha_{gw} \cdot SDP_{OPT}$, resulting in an approximation ratio of at least .87856 for MAX-CUT. The same analysis applies to weighted graphs with non-negative edge weights.

This algorithm was de-randomized by Mahajan and Hariharan [23]. Goemans and Williamson also applied their random-hyperplane rounding techniques to give improved approximation guarantees for other problems such as MAX-DICUT and MAX-2SAT.

Integrality Gap and Hardness

Karloff showed that there exist graphs for which the best hyperplane is only a factor α_{gw} of the maximum cut [18], showing that there are graphs for which the analysis in [11] is tight. Since the optimal SDP value for such graphs equals the optimal value of a maximum cut, these graphs can not be used to demonstrate an integrality gap. However, Feige and Schechtman showed that there exist graphs for which the maximum cut is a α_{gw} fraction of the SDP bound [9], thereby establishing that the approximation guarantee of Goemans and Williamson's algorithm matches the integrality gap of their semidefinite programming relaxation. Recently, Khot, Kindler, Mossel and O'Donnell [21] showed that if the Unique Games Conjecture of Khot [20] is assumed to be true, then it is NP-hard to approximate MAX-CUT to within any factor larger than α_{gw}.

Applications

The work of Goemans and Williamson paved the way for the further use of semidefinite programming in approximation algorithms, particularly for graph partitioning problems. Methods based on the random-hyperplane technique have been successfully applied to many optimization problems that can be categorized as partition problems. A few examples are 3-COLORING [17], MAX-3-CUT [10,12,22], MAX-BISECTION [15], CORRELATION-CLUSTERING [5,32], and SPARSEST-CUT [2]. Additionally, some progress has been made in extending semidefinite programming techniques outside the domain of graph partitioning to problems such as BETWEENNESS [6], BANDWIDTH [4], and LINEAR EQUATIONS mod p [1].

Cross References

► Graph Coloring
► Maximum Two-Satisfiability
► Sparsest Cut

Recommended Reading

1. Andersson, G., Engebretsen, L., Håstad, J.: A new way to use semidefinite programming with applications to linear equations mod *p*. J. Algorithms **39**, 162–204 (2001)
2. Arora, S., Rao, S., Vazirani, U.: Expander flows, geometric embeddings and graph partitioning. In: Proceedings of the 36th Annual Symposium on the Theory of Computing (STOC), Chicago 2004, pp. 222–231
3. Barahona, F.: On cuts and matchings in planar graphs. Math. Program. **60**, 53–68 (1993)
4. Blum, A., Konjevod, G., Ravi, R., Vempala, S.: Semi-definite relaxations for minimum bandwidth and other vertex-ordering problems. Theor. Comput. Sci. **235**, 25–42 (2000)
5. Charikar, M., Guruswami, V., Wirth, A.: Clustering with qualitative information. In: Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS), Boston 2003, pp. 524–533
6. Chor, B., Sudan, M.: A geometric approach to betweeness. SIAM J. Discret. Math. **11**, 511–523 (1998)
7. Delorme, C., Poljak, S.: Laplacian eigenvalues and the maximum cut problem. Math. Program. **62**, 557–574 (1993)
8. Delorme, C., Poljak, S.: The performance of an eigenvalue bound in some classes of graphs. Discret. Math. **111**, 145–156 (1993). Also appeared in: Proceedings of the Conference on Combinatorics, Marseille, 1990
9. Feige, U., Schechtman, G.: On the optimality of the random hyperplane rounding technique for MAX-CUT. Random Struct. Algorithms **20**(3), 403–440 (2002)
10. Frieze, A., Jerrum, M.R.: Improved approximation algorithms for MAX-*k*-CUT and MAX BISECTION. Algorithmica **18**, 61–77 (1997)
11. Goemans, M.X., Williamson, D.P.: Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. J. ACM **42**, 1115–1145 (1995)
12. Goemans, M.X., Williamson, D.P.: Approximation algorithms for MAX-3-CUT and other problems via complex semidefinite programming. STOC 2001 Special Issue of J. Comput. Syst. Sci. **68**, 442–470 (2004)
13. Grötschel, M., Lovász, L., Schrijver, A.: The ellipsoid method and its consequences in combinatorial optimization. Combinatorica **1**, 169–197 (1981)
14. Grötschel, M., Lovász, L., Schrijver, A.: Geometric Algorithms and Combinatorial Optimization. Springer, Berlin (1988)
15. Halperin, E., Zwick, U.: A unified framework for obtaining improved approximation algorithms for maximum graph bisection problems. Random Struct. Algorithms **20**(3), 382–402 (2002)
16. Håstad, J.: Some optimal inapproximability results. J. ACM **48**, 798–869 (2001)
17. Karger, D.R., Motwani, R., Sudan, M.: Improved graph coloring via semidefinite programming. J. ACM **45**(2), 246–265 (1998)
18. Karloff, H.J.: How good is the Goemans-Williamson MAX CUT algorithm? SIAM J. Comput. **29**(1), 336–350 (1999)
19. Karp, R.M.: Reducibility Among Combinatorial Problems. In: Complexity of Computer Computations, pp. 85–104. Plenum Press, New York (1972)
20. Khot, S.: On the power of unique 2-prover 1-round games. In: Proceedings of the 34th Annual Symposium on the Theory of Computing (STOC), Montreal 2002, pp. 767–775
21. Khot, S., Kindler, G., Mossel, E., O'Donnell, R.: Optimal inapproximability results for MAX CUT and other 2-variable CSPs? In: Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS), Rome 2004, pp. 146–154
22. de Klerk, E., Pasechnik, D., Warners, J.: On approximate graph colouring and MAX-*k*-CUT algorithms based on the θ function. J. Combin. Optim. **8**(3), 267–294 (2004)
23. Mahajan, R., Hariharan, R.: Derandomizing semidefinite programming based approximation algorithms. In: Proceedings of the 36th Annual IEEE Symposium on Foundations of Computer Science (FOCS), Milwaukee 1995, pp. 162–169
24. Mohar, B., Poljak, S.: Eigenvalues and the max-cut problem. Czechoslov Math. J. **40**(115), 343–352 (1990)
25. Newman, A.: A note on polyhedral relaxations for the maximum cut problem (2004). Unpublished manuscript
26. Poljak, S.: Polyhedral and eigenvalue approximations of the max-cut problem. Sets, Graphs and Numbers. Colloqiua Mathematica Societatis Janos Bolyai **60**, 569–581 (1992)
27. Poljak, S., Rendl, F.: Node and edge relaxations of the max-cut problem. Comput. **52**, 123–137 (1994)
28. Poljak, S., Rendl, F.: Nonpolyhedral relaxations of graph-bisection problems. SIAM J. Opt. **5**, 467–487 (1995)
29. Poljak, S., Rendl, F.: Solving the max-cut using eigenvalues. Discret. Appl. Math. **62**(1–3), 249–278 (1995)
30. Poljak, S., Tuza, Z.: Maximum cuts and large bipartite subgraphs. DIMACS Ser. Discret. Math. Theor. Comput. Sci. **20**, 181–244 (1995)
31. Sahni, S., Gonzalez, T.: P-complete approximation problems. J. ACM **23**(3), 555–565 (1976)
32. Swamy, C.: Correlation clustering: maximizing agreements via semidefinite programming. In: Proceedings of 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), New Orleans 2004, pp. 526–527
33. Trevisan, L., Sorkin, G., Sudan, M., Williamson, D.: Gadgets, approximation, and linear programming. SIAM J. Comput. **29**(6), 2074–2097 (2000)

Maximum Agreement Subtree (of 2 Binary Trees)

1996; Cole, Hariharan

RAMESH HARIHARAN
Strand Life Sciences, Bangalore, India

Keywords and Synonyms

Isomorphism; Tree agreement

Problem Definition

Consider two rooted trees T_1 and T_2 with n leaves each. The internal nodes of each tree have at least two children each. The leaves in each tree are labeled with the same set of labels and further, no label occurs more than once in a particular tree. An *agreement subtree* of T_1 and T_2 is defined as follows. Let L_1 be a subset of the leaves of T_1 and let L_2 be the subset of those leaves of T_2 which have the same labels as leaves in L_1. The subtree of T_1 *induced* by L_1 is an agreement subtree of T_1 and T_2 if and only if it is *isomorphic* to the subtree of T_2 induced by L_2. The Maximum Agreement Subtree problem (henceforth called *MAST*) asks for the largest agreement subtree of T_1 and T_2.

The terms *induced subtree* and *isomorphism* used above need to be defined. Intuitively, the subtree of T induced by a subset L of the leaves of T is the topological subtree of T restricted to the leaves in L, with branching information relevant to L preserved. More formally, for any two leaves $a\,b$ of a tree T, let $\mathrm{lca}_T(a,b)$ denote their lowest common ancestor in T. If $a = b$, $\mathrm{lca}_T(a,b) = a$. The *subtree U of T induced* by a subset L of the leaves is the tree with leaf set L and interior node set $\{\mathrm{lca}_T(a,b) | a, b \in L\}$ inheriting the ancestor relation from T, that is, for all $a, b \in L$, $\mathrm{lca}_U(a,b) = \mathrm{lca}_T(a,b)$.

Intuitively, two trees are isomorphic if the children of each node in one of the trees can be reordered so that the leaf labels in each tree occur in the same order and the shapes of the two trees become identical. Formally, two trees U_1 and U_2 with the same leaf labels are said to be *isomorphic* if there is a 1–1 mapping μ between their nodes mapping leaves to leaves with the same labels and such that for any two different leaves $a\,b$ of U_1, $\mu(\mathrm{lca}_{U_1}(a,b)) = \mathrm{lca}_{U_2}(\mu(a), \mu(b))$.

Key Results

Previous Work

Finden and Gordon [8] gave a heuristic algorithm for the *MAST* problem on binary trees which had an $O(n^5)$ running time and did not guarantee an optimal solution. Kubicka, Kubicki and McMorris [13] gave an $O(n^{(.5+\epsilon)\log n})$ algorithm for the same problem. The first polynomial time algorithm for this problem was given by Steel and Warnow [15]; it had a running time of $O(n^2)$. Steel and Warnow also considered the case of non-binary and unrooted trees. Their algorithm takes $O(n^2)$ time for fixed degree rooted and unrooted trees and $O(n^{4.5}\log n)$ for arbitrary degree rooted and unrooted trees. They also give a linear reduction from the rooted to the unrooted case. Farach and Thorup gave an $O(nc^{\sqrt{\log n}})$ time algorithm for the *MAST* problem on binary trees; here c is a constant greater than 1. For arbitrary degree trees, their algorithm takes $O(n^2 c^{\sqrt{\log n}})$ time for the unrooted case [6] and $O(n^{1.5}\log n)$ time for the rooted case [7]. Farach, Przytycka, and Thorup [4] obtained an $O(n\log^3 n)$ algorithm for the *MAST* problem on binary trees. Kao [12] obtained an algorithm for the same problem which takes $O(n\log^2 n)$ time. This algorithm takes $O(\min\{nd^2\log d\log^2 n, nd^{\frac{3}{2}}\log^3 n\})$ for degree d trees.

The *MAST* problem for more than two trees has also been studied. Amir and Keselman [1] showed that the problem is *NP*-hard for even 3 unbounded degree trees. However, polynomial bounds are known [1,5] for three or more bounded degree trees.

Our Contribution

An $O(n\log n)$ algorithm for the *MAST* problem for two binary trees is presented here. This algorithms is obtained by improving upon the $O(n\log^3 n)$ algorithm from [4] (in fact, the final journal version [3] combines both papers). The $O(n\log^3 n)$ algorithm of [4] can be viewed as taking the following approach (although the authors do not describe it this way). It identifies two special cases and then solves the general case by interpolating between these cases.

Special Case 1: The internal nodes in both trees form a path. The *MAST* problem reduces to essentially a size n Longest Increasing Subsequence Problem in this case. As is well known, this can be solved in $O(n\log n)$ time.

Special Case 2: Both trees T_1 and T_2 are complete binary trees. For each node v in T_2, only certain nodes u in T_1 can be usefully mapped to v, in the sense that the subtree of T_1 rooted at u and the subtree of T_2 rooted at v have a non-empty Agreement Subtree. There are $O(n\log^2 n)$ such pairs (u, v). This can be seen as follows. Note that for (u, v) to be such a pair, the subtree of T_1 rooted at u and the subtree of T_2 rooted at v must have a leaf-label in common. For each label, there are only $O(\log^2 n)$ such pairs obtained by pairing each ancestor of the leaf with this label in T_1 with each ancestor of the leaf with this label in T_2. The total number of interesting pairs is thus $O(n\log^2 n)$. For each pair, computing the *MAST* takes $O(1)$ time, as it is simply a question of deciding the best way of pairing their children.

The interpolation process takes a centroid decomposition of the two trees and compares pairs of centroid paths, rather than individual nodes as in the complete tree case. The comparison of a pair of centroid paths requires finding matchings with special properties in appropriately defined bipartite graphs, a non-trivial generalization of the Longest Increasing Subsequence problem. This process creates $O(n \log^2 n)$ interesting (u, v) pairs, each of which takes $O(\log n)$ time to process.

This work provides two improvements, each of which gains a $\log n$ factor.

Improvement 1: The complete tree special case is improved to $O(n \log n)$ time as follows. A pair of nodes (u, v), $u \in T_1$, $v \in T_2$, is said to be *interesting* if there is an agreement subtree mapping u to v. As is shown below, for complete trees, the total number of interesting pairs (u, v) is just $O(n \log n)$. Consider a node v in T_2. Let L_2 be the set of leaves which are descendants of v. Let L_1 be the set of leaves in T_1 which have the same labels as the leaves in L_2. The only nodes that may be mapped to v are the nodes u in the subtree of T_1 induced by L_1. The number of such nodes u is O(size of the subtree of T_2 rooted at v). The total number of interesting pairs is thus the sum of the sizes of all subtrees of T_2, which is $O(n \log n)$.

This reduces the number of interesting pairs (u, v) to $O(n \log n)$. Again, processing a pair takes $O(1)$ time (this is less obvious, for identifying the descendants of u which root the subtrees with which the two subtrees of v can be matched is non-trivial). Constructing the above induced subtree itself can be done in $O(|L_1|)$ time, as will be detailed later. The basic tool here is to preprocess trees T_1 and T_2 in $O(n)$ time so that least common ancestor queries can be answered in $O(1)$ time.

Improvement 2: As in [4], when the trees are not complete binary trees, the algorithm takes centroid paths and matches pairs of centroid paths. The $O(\log n)$ cost that the algorithm in [4] incurs in processing each such interesting pair of paths arises when there are large (polynomial in n size) instances of the generalized Longest Increasing Subsequence Problem. At first sight, it is not clear that large instances of these problems can be created for sufficiently many of the interesting pairs; unfortunately, this turns out to be the case. However, these problem instances still have some useful structure. By using (static) weighted trees, pairs of interesting vertices are processed in $O(1)$ time per pair, on the average, as is shown by an appropriately parametrized analysis.

The Multiple Degree Case

The techniques can be generalized to higher degree bounds $d > 2$, by combining it with techniques from ([6, Sect. 2]) for unbounded degrees. This appears to yield an algorithm with running time $O(\min\{n\sqrt{d}\log^2 n, nd \log n \log d\})$. The conjecture, however, is that there is an algorithm with running time $O(n\sqrt{d}\log n)$.

Applications

Motivation

The *MAST* problem arises naturally in biology and linguistics as a measure of consistency between two evolutionary trees over species and languages, respectively. An evolutionary tree for a set of *taxa*, either species or languages, is a rooted tree whose leaves represent the taxa and whose internal nodes represent ancestor information. It is often difficult to determine the true phylogeny for a set of taxa, and one way to gain confidence in a particular tree is to have different lines of evidence supporting that tree. In the biological taxa case, one may construct trees from different parts of the DNA of the species. These are known as *gene trees*. For many reasons, these trees need not entirely agree, and so one is left with the task of finding a consensus of the various gene trees. The maximum agreement subtree is one method of arriving at such a consensus. Notice that a gene is usually a binary tree, since DNA replicates by a binary branching process. Therefore, the case of binary trees is of great interest.

Another application arises in automated translation between two languages [10]. The two trees are the parse trees for the same meaning sentences in the two languages. A complication that arises in this application (due in part to imperfect dictionaries) is that words need not be uniquely matched, i. e., a word at the leaf of one tree could match a number (usually small) of words at the leaves of the other tree. The aim is to find a maximum agreement subtree; this is done with the goal of improving context-using dictionaries for automated translation. So long as each word in one tree has only a constant number of matches in the other tree (possibly with differing weights), the algorithm given here can be used and its performance remains $O(n \log n)$. More generally, if there are m word matches in all, the performance becomes $O((m + n) \log n)$. Note however, that if there are two collections of equal meaning words in the two trees of sizes k_1 and k_2 respectively, they induce $k_1 k_2$ matches.

Cross References

► Maximum Agreement Subtree (of 3 or More Trees)
► Maximum Agreement Supertree

Recommended Reading

1. Amir, A., Keselman, D.: Maximum agreement subtree in a set of evolutionary trees. SIAM J. Comput. **26**(6), 1656–1669 (1997)
2. Cole, R., Hariharan, R.: An $O(n \log n)$ algorithm for the maximum agreement subtree problem for binary trees. Proc. of the 7th ACM-SIAM SODA, pp. 323–332 (1996)

3. Cole, R., Farach-Colton, M., Hariharan, R., Przytycka, T., Thorup, M.: An $O(n \log n)$ algorithm for the maximum agreement subtree problem for binary trees. SIAM J. Comput. **30**(5), 1385–1404 (2000)
4. Farach, M., Przytycka, T., Thorup, M.: The maximum agreement subtree problem for binary trees. Proc. of 2nd ESA (1995)
5. Farach, M., Przytycka, T., Thorup, M.: Agreement of many bounded degree evolutionary trees. Inf. Process. Lett. **55**(6), 297–301 (1995)
6. Farach, M., Thorup, M.: Fast comparison of evolutionary trees. Inf. Comput. **123**(1), 29–37 (1995)
7. Farach, M., Thorup, M.: Sparse dynamic programming for evolutionary-tree comparison. SIAM J. Comput. **26**(1), 210–230 (1997)
8. Finden, C.R., Gordon, A.D.: Obtaining common pruned trees. J. Classific. **2**, 255–276 (1985)
9. Fredman, M.L.: Two applications of a probabilistic search technique: sorting $X + Y$ and building balanced search trees. Proc. of the 7th ACM STOC, pp. 240–244 (1975)
10. Grishman, R., Yangarber, R.: Private Communication. NYU (1995)
11. Harel, D., Tarjan, R.E.: Fast algorithms for finding nearest common ancestors. SIAM J. Comput. **13**(2), 338–355 (1984)
12. Kao, M.-Y.: Tree contractions and evolutionary trees. SIAM J. Comput. **27**(6), 1592–1616 (1998)
13. Kubicka, E., Kubicki, G., McMorris, F.R.: An algorithm to find agreement subtrees. J. Classific. **12**, 91–100 (1995)
14. Mehlhorn, K.: A best possible bound for the weighted path length of binary search trees. SIAM J. Comput. **6**(2), 235–239 (1977)
15. Steel, M., Warnow, T.: Kaikoura tree theorems: computing the maximum agreement subtree. Inf. Process. Lett. **48**, 77–82 (1993)

Maximum Agreement Subtree (of 3 or More Trees)

1995; Farach, Przytycka, Thorup

Teresa M. Przytycka
Computational Biology Branch, NCBI, NIH,
Bethesda, MD, USA

Keywords and Synonyms

Tree alignment

Problem Definition

The Maximum Agreement Subtree problem for k trees (k-MAST) is a generalization of a similar problem for two trees (MAST). Consider a tuple of k rooted leaf-labeled trees $(T_1, T_2 \ldots T_k)$. Let $A = \{a_1, a_2, \ldots a_n\}$ be the set of leaf labels. Any subset $B \subseteq A$ uniquely determines the so called *topological restriction* $T|B$ of the three T to B. Namely, $T|B$ is the topological subtree of T spanned by all leaves labeled with elements from B and lowest common ancestors of all pairs of these leaves. In particular, the ancestor relation in $T|B$ is defined so that it agrees with the ancestor relation in T. A subset B of A such $T^1|B, \ldots, T^k|B$ are isomorphic is called an *agreement set*.

Problem 1 (k-MAST)
INPUT: *A tuple $\vec{T} = (T^1, \ldots, T^k)$ of leaf-labeled trees, with a common set of labels $A = \{a_1, \ldots, a_n\}$, such that for each tree T^i there exists one-to-one mapping between the set of leaves of that tree and the set of labels A.*
OUTPUT: *k-MAST($\vec{T}$) equal to the maximum cardinality agreement set of $\vec{T}$.*

Key Results

In the general setting, k-MAST problem is NP-complete for $k \geq 3$ [1]. Under the assumption that the degree of at least one of the trees is bounded, Farach et al. proposed an algorithm leading to the following theorem:

Theorem 1 *If the degree of the trees in the tuple $\vec{T} = (T^1, \ldots, T^k)$ is bounded by d then the k-MAST($\vec{T}$) can be computed in $O(kn^3 + n^d)$ time.*

In what follows, the problem is restricted to finding the cardinality of the maximum agreement set rather than the set itself. The extension of this algorithm to an algorithm that finds the agreement set (and subsequently the agreement subtree) within the same time bounds is relatively straightforward.

Recall that the classical $O(n^2)$ dynamic programming algorithm for MAST of two binary trees [11] processes all pairs of internal nodes of the two trees in a bottom up fashion. For each pair of such nodes it computes the MAST value for the subtrees rooted at this pair. There are $O(n^2)$ pairs of nodes and the MAST value for the subtrees rooted at a given pair of nodes can be computed in constant time from MAST values of previously processed pairs of nodes.

To set the stage for the more general case, let k-MAST($\vec{v}$) be the solution to the k-MAST problem for the subtrees of $T^1(v_1), \ldots, T^k(v_k)$ where $T^i(v_i)$ is the subtree if T^i rooted at v_i. If, for all i, u_i is a strict ancestor of v_i in T^i then, $\vec{v}$ is *dominated* by $\vec{u}$ (denoted $\vec{v} \prec \vec{u}$).

A naive extension of the algorithm for two trees to an algorithm for k trees would require computing k-MAST($\vec{v}$) for all possible tuples $\vec{v}$ by processing these tuples in the order consistent with the domination relation. The basic idea that allows to avoid $\Omega(n^k)$ complexity is to replace the computation of k-MAST($\vec{v}$) with the computation of a related value, mast($\vec{v}$), defined to be the size of the maximum agreement set for the subtrees of $(T^1, \ldots, T^k)$ rooted at $(v_1, \ldots v_k)$ subject to the additional restriction that the agreement subtrees themselves are rooted at $v_1, \ldots v_k$ respectively. Clearly

$$\text{k-MAST}(T^1, \ldots, T^k) = \max_{\vec{v}} \text{mast}(\vec{v}) .$$

The benefit of computing mast rather than k-MAST follows from the fact that most of mast values are zero and it is possible to identify (very efficiently) $\vec{v}$ with non-zero mast values.

Remark 1 If $\text{mast}(\vec{v}) > 0$ then $\vec{v} = (\text{lca}^{T^1}(a, b), \ldots \text{lca}^{T^k}(a, b))$ for some leaf labels a, b where $\text{lca}^{T^i}(a, b)$ is the lowest common ancestor of leaves labeled by a and b in the tree T^i.

A tuple $\vec{v}$ such that $\vec{v} = (\text{lca}^{T^1}(a, b), \ldots \text{lca}^{T^k}(a, b))$ for some $a, b \in A$ is called an *lca-tuple*. By Remark 1 it suffices to compute mast values for the lca-tuples only. Just like in the naive approach, $\text{mast}(\vec{v})$ is computed from mast values of other lca-tuples dominated by $\vec{v}$. Another important observation is that only some lca-tuples dominated by $\vec{v}$ are needed to compute $\text{mast}(\vec{v})$. To capture this, Farach et al. define the so called *proper domination* relation (introduced formally below) and show that the mast value for any lca-tuple $\vec{v}$ can be computed from mast values of lca-tuples properly dominated by $\vec{v}$ and that the proper domination relation has size $O(n^3)$.

Proper Domination Relation

Index the children of each internal node of any tree in an arbitrary way. Given a pair $\vec{v}, \vec{w}$ of lca-tuples such that $\vec{w} \prec \vec{v}$ the corresponding domination relation has associated with it *direction* $\vec{\delta}_{\vec{w} \prec \vec{v}} = (\delta_1, \ldots, \delta_k)$ where w_i descends from the child of v_i indexed with δ_i. Let $v_i(j)$ be the child of v_i with index j. The direction domination is termed *active* is if the subtrees rooted at the $v_1(\delta_1), \ldots, v_k(\delta_k)$ have at least one leaf label in common. Note that each leaf label can witness only one active direction, and consequently each lca-tuple can have at most n active domination directions. Two directions $\vec{\delta}_{\vec{w} \prec \vec{v}}$ and $\vec{\delta}_{\vec{u} \prec \vec{v}}$ are called *compatible* if and only if the direction vectors differ in all coordinates.

Definition 1 $\vec{v}$ properly denominates $\vec{u}$ (denoted $\vec{u} < \vec{v}$) if $\vec{v}$ dominates $\vec{u}$ along an active direction $\vec{\delta}$ and there exists another tuple $\vec{w}$ which is also dominated by $\vec{v}$ along an active direction $\vec{\delta}_\perp$ compatible with δ.

From the definition of proper domination and from the fact that each leaf label can witness only one active domination direction, the following observations can be made:

Remark 2 The strong domination relation $<$ on lca-tuples has size $O(n^3)$. Furthermore, the relation can be computed in $O(kn^3)$ time.

Remark 3 For any lca-tuple $\vec{v}$, if $\text{mast}(\vec{v}) > 0$ then either $\vec{v}$ is an lca-tuple composed of leaves with the same label or $\vec{v}$ properly dominates some lca-tuple.

It remains to show how the values $\text{mast}(\vec{v})$ are computed. For each lca-tuple $\vec{v}$, the so called compatibility graph $G(\vec{v})$ is constructed. The nodes of $G(\vec{v})$ are active directions from $\vec{v}$ and there is an edge between two such nodes if and only if corresponding directions are compatible. The vertices of $G(\vec{v})$ are weighted and the weight of a vertex corresponding to an active direction $\vec{\delta}$ equals the maximum mast value of a lca-tuple dominated by $\vec{v}$ along the this direction. Let $MWC(G(\vec{v}))$ be the maximum weight clique in $G(\vec{v})$.

The bottom-up algorithm for computing non-zero mast values based on the following recursive dependency whose correctness follows immediately from the corresponding definitions and Remark 3:

Lemma 2 *For any lca-tuple* $\vec{v}$

$$\text{mast}(\vec{v}) = \max \begin{cases} 1 & \text{if all elemets of } \vec{v} \text{ are leaves} \\ MWC(G(\vec{v})) & \text{otherwise} \end{cases} . \quad (1)$$

The final step is to demonstrate that once the lca-tuples and the strong domination relation is pre-computed, the computation all non-zero mast values can be preformed in $O(n^d)$ time. This is done by generating all possible cliques for all $G(\vec{v})$. Using the fact that the degree of at least one tree is bounded by d one can show that all the cliques can be generated in $O(\sum_{l \leq d} \binom{n}{l}) = O(d^3(ne/d)^d)$ time and that there is $O(d(ne/d)^d)$ of them [6].

Applications

The k-MAST problem is motivated by the need to compare evolutionary trees. Recent advances in experimental techniques in molecular biology provide diverse data that can be used to construct evolutionary trees. This diversity of data combined with the diversity of methods used to construct evolutionary trees often leads to the situation when the evolution of the same set of species is explained by different evolutionary trees. Maximum Agreement Subtree problem has emerged as a measure of the agreement between such trees and as a method to identify subtree which is common for these trees. The problem was first defined by Finden and Gordon in the context of two binary trees [7]. These authors also gave a heuristic algorithm to solve the problem. The $O(n^2)$ dynamic programming algorithm for computing MAST values for two binary trees has been given in [11]. Subsequently, a number of improvements leading to fast algorithms for computing MAST value of two trees under various assumption about rooting and tree degrees [5,10,8] and references therein.

The MAST problem for three or more unbounded degree trees is NP-complete [1]. Amir and Keselman report an $O(kn^{d+1} + n^{2d})$ time algorithm for the agreement of k bounded degree trees. The work described here provides a $O(kn^3 + n^d)$ for the case where the number of trees is k and the degree of at least one tree is bounded by d. For $d = 2$ the complexity of the algorithm is dominated by the first term. An $O(kn^3)$ algorithm for this case was also given by Bryant [4] and $O(n^2 \log^{k-1} n)$ implementation of this algorithm was proposed in [9].

k-MAST problem is fixed parameter tractable in p, the smallest number of leafs labels such that removal of the corresponding leaves produces agreement (see [3] and references therein). Approximability of the MAST and related problem has been studied in [2] and references therein.

Cross References

- ▶ Maximum Agreement Subtree (of 2 Binary Trees)
- ▶ Maximum Agreement Supertree
- ▶ Maximum Compatible Tree

Acknowledgments

This work was supported by the Intramural Research Program of the National Institutes of Health, National Library of Medicine.

Recommended Reading

1. Amir, A., Keselman, D.: Maximum agreement subtree in a set of evolutionary trees: Metrics and efficient algorithms. SIAM J. Comput. **26**(6), 1656–1669 (1997)
2. Berry, V., Guillemot, S., Nicolas, F., Paul, C.: On the approximation of computing evolutionary trees. In: COCOON, pp. 115–125. (2005)
3. Berry, V., Nicolas, F.: Improved parameterized complexity of the maximum agreement subtree and maximum compatible tree problems. IEEE/ACM Trans. Comput. Biology Bioinform. **3**(3), 289–302 (2006)
4. Bryand, D.: Building trees, hunting for trees, and comparing trees: theory and methods in phylogenetic analysis. In: Ph. D. thesis, Dept. Math., University of Canterbury (1997)
5. Cole, R., Farach-Colton, M., Hariharan, R., Przytycka, T., Thorup, M.: An $o(n \log n)$ algorithm for the maximum agreement subtree problem for binary trees. SIAM J. Comput., pp. 1385–1404. (2001)
6. Farach, M., Przytycka, T.M., Thorup, M.: On the agreement of many trees. Inf. Process. Lett. **55**(6), 297–301 (1995)
7. Finden, C.R., Gordon, A.D.: Obtaining common pruned trees. J. Classific. **2**, 255–276 (1985)
8. Kao, M.-Y., Lam, T.-W., Sung, W.-K., Ting, H.-F.: An even faster and more unifying algorithm for comparing trees via unbalanced bipartite matchings. J. Algorithms **40**(2), 212–233 (2001)
9. Lee, C.-M., Hung, L.-J., Chang, M.-S., Tang, C.-Y.: An improved algorithm for the maximum agreement subtree problem. BIBE, p. 533 (2004)
10. Przytycka, T.M.: Transforming rooted agreement into unrooted agreement. J. Comput. Biol. **5**(2), 335–349 (1998)
11. Steel, M.A., Warnow, T.: Kaikoura tree theorems: Computing the maximum agreement subtree. Inf. Process. Lett. **48**(2), 77–82 (1993)

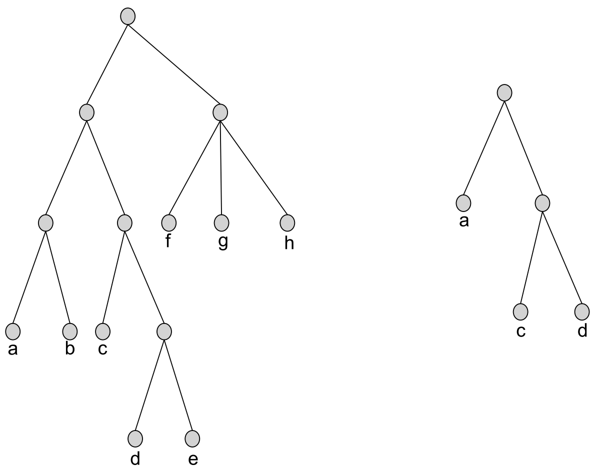
Maximum Agreement Supertree

2005; Jansson, Ng, Sadakane, Sung

WING-KIN SUNG
Department of Computer Science, National University of Singapore, Singapore, Singapore

Problem Definition

Let T be a tree whose leaves are distinctly labeled by a set of taxa S. By distinctly labeled, we mean that no two leaves in T have the same label. Given a subset S' of S, *the topological restriction of T to S'* (denoted by $T|S'$) is the tree obtained by deleting from T all nodes which are not on any path from the root to a leaf in S' along with their incident edges, and then contracting every edge between a node having just one child and its child (see Fig. 1). For any tree T, denote its set of leaves by $\Lambda(T)$.



Maximum Agreement Supertree, Figure 1
Let T be the tree on the *left*. Then $T|\{a, c, d\}$ is the tree shown on the *right*

The maximum agreement supertree problem (MASP) [8] is defined as follows.

Problem 1 *Let $D = \{T_1, T_2, \dots, T_k\}$ be a set of rooted, unordered trees, where each T_i is distinctly leaf-labeled and where the sets $\Lambda(T_i)$ may overlap. The maximum agreement supertree problem (MASP) is to construct a distinctly leaf-labeled tree Q with leaf set $\Lambda(Q) \subseteq \bigcup_{T_i \in D} \Lambda(T_i)$ such that $|\Lambda(Q)|$ is maximized and for each $T_i \in D$, the topological restriction of T_i to $\Lambda(Q)$ is isomorphic to the topological restriction of Q to $\Lambda(T_i)$.*

Below discussion uses the following notations: $n = \left|\bigcup_{T_i \in D} \Lambda(T_i)\right|$, $k = |D|$, and $D = \max_{T_i \in D}\{\deg(T_i)\}$ where $deg(T_i)$ is the degree of T_i.

Key Results

The following lemma gives the relationship between the maximum agreement supertree problem and the maximum agreement subtree problem.

Lemma 1 ([8]) *For any set $D = \{T_1, T_2, \dots, T_k\}$ of distinctly leaf-labeled, rooted, unordered trees such that $\Lambda(T_1) = \Lambda(T_2) = \dots = \Lambda(T_k)$, an optimal solution to MASP for D is an optimal solution to MAST for D and vice versa.*

The above lemma implies the following theorem for computing the maximum agreement supertree for two trees.

Theorem 2 ([8]) *When $k = 2$ (there are two trees), the maximum agreement supertree can be found in $O(T_{MAST} + n)$ time where T_{MAST} is the time required for computing maximum agreement subtree of two $O(n)$-leaf trees. Note that $T_{MAST} = O\left(\sqrt{D}n \log(2n/D)\right)$ (see [9]).*

[1] generalized Theorem 2 and gave the following solution.

Theorem 3 ([1]) *For any fixed $k > 2$, if every leaf in D appears in either 1 or k trees, the maximum agreement supertree can be found in $O(T'_{MAST} + kn)$ time where T'_{MAST} is the time required for computing maximum agreement subtree of k trees leaf-labeled by $\bigcap_{T_i \in D} \Lambda(T_i)$. Note that $T'_{MAST} = O(km^3 + m^D)$ where $n = \left|\bigcap_{T_i \in D} \Lambda(T_i)\right|$ (see [4]).*

In general, the following two theorems showed that the maximum agreement supertree problem is NP-hard.

Theorem 4 ([8,1]) *For any fixed $k \geq 3$, MASP with unrestricted D is NP-hard. Even stronger, MASP is still NP-hard even if restricted to rooted triplets. (A rooted triplet is a distinctly leaf-labeled, binary, rooted, unordered tree with three leaves.)*

Theorem 5 ([1]) *MASP cannot be approximated in polynomial time within a constant factor, unless $P = NP$.*

Though the MASP problem is NP-hard, approximation algorithm for this problem exists.

Theorem 6 ([8]) *MASP can be approximated within a factor of $\frac{n}{\log n}$ in $O(n^2) \cdot \min\left\{O(k \cdot (\log\log n)^2),\ O(k + \log n \cdot \log\log n)\right\}$ time. MASP restricted to rooted triplets can be approximated within a factor of $\frac{n}{\log n}$ in $O(k + n^2 \log^2 n)$ time.*

Fixed parameter polynomial time algorithms for computing MASP also exist. For the case where a set of k binary trees T labeled by n distinct labels is given, a number of works have been done. Jansson et al.[8] first gave an $O(k(2n^2)^{3k^2})$-time algorithm to compute the MASP of T. Recently, Guillemot and Berry [5] improved the time complexity to $O((8n)^k)$. Hoang and Sung [7] further improved the time complexity to $O((6n)^k)$ as summarized by Theorem 7.

Theorem 7 ([7]) *Given a set of k binary trees T which are labeled by n distinct labels, their maximum agreement supertree can be computed in $O((6n)^k)$ time.*

For the case where a set of k trees T are of degree D and are labeled by n distinct labels, Hoang and Sung [7] gave the following fixed-parameter polynomial time solution to compute the MASP of T.

Theorem 8 ([7]) *Given a set of k trees T of degree D which are labeled by n distinct labels, their maximum agreement supertree can be computed in $O((kD)^{kD+3}(2n)^k)$ time.*

Applications

An important objective in phylogenetics is to develop good methods for merging a collection of phylogenetic trees on overlapping sets of taxa into a single supertree so that no (or as little as possible) branching information is lost. Ideally, the resulting supertree can then be used to deduce evolutionary relationships between taxa which do not occur together in any one of the input trees. Supertree methods are useful because most individual studies investigate relatively few taxa [11] and because sample bias leads to certain taxa being studied much more frequently than others [2]. Also, supertree methods can combine trees constructed for different types of data or under different models of evolution. Furthermore, although computationally expensive methods for constructing reliable phylogenetic trees are infeasible for large sets of taxa, they can be applied to obtain highly accurate trees for smaller, overlapping subsets of the taxa which may then be merged using computationally less intense, supertree-based techniques (see, e. g., [3,6,10]).

Since the set of trees which is to be combined may in practice contain contradictory branching structure (for example, if the trees have been constructed from data originating from different genes or if the experimental data contains errors), a supertree method needs to specify how to resolve conflicts. One intuitive idea is to identify and remove a smallest possible subset of the taxa so that the remaining taxa can be combined without conflicts. In this way, one would get an indication of which ancestral relationships can be regarded as resolved and which taxa need to be subjected to further experiments. The above biological problem can be formalized as a computational problem called the maximum agreement supertree problem (MASP).

A related problem is the maximum compatible supertree problem (MCSP) [1], which is defined as follows.

Problem 2 *Let* $D = \{T_1, T_2, \ldots, T_k\}$ *be a set of rooted, unordered trees, where each* T_i *is distinctly leaf-labeled and where the sets* $\Lambda(T_i)$ *may overlap. The maximum compatible supertree problem (MCSP) is to construct a distinctly leaf-labeled tree Q with leaf set* $\Lambda(Q) \subseteq \bigcup_{T_i \in D} \Lambda(T_i)$ *such that* $|\Lambda(Q)|$ *is maximized and for each* $T_i \in D$*, The topological restriction* $Q_i{}'$ *of Q to* $\Lambda(T_i)$ *refines the topological restriction* $T_i{}'$ *of* T_i*, that is,* $T_i{}'$ *can be obtained by collapsing certain edges of* $Q_i{}'$*.*

Open Problems

The current fixed parameter polynomial time algorithms for MASP are not practical. It is important to provide heuristics or to further improve the time complexity of current fixed-parameter polynomial time algorithms.

Cross References

▶ Maximum Agreement Subtree (of 2 Binary Trees)
▶ Maximum Agreement Subtree (of 3 or More Trees)
▶ Maximum Compatible Tree

Recommended Reading

1. Berry, V., Nicolas, F.: Maximum agreement and compatible supertrees. J. Discret. Algorithms (2006)
2. Bininda-Emonds, O., Gittleman, J., Steel, M.: The (super)tree of life: Procedures, problems, and prospects. Ann. Rev. Ecol. System. **33**, 265–289 (2002)
3. Chor, B., Hendy, M., Penny, D.: Analytic solutions for three-taxon ML_{MC} trees with variable rates across sites. In: Proceedings of the 1st Workshop on Algorithms in Bioinformatics (WABI 2001). Lecture Notes in Computer Science, vol. 2149, pp. 204–213. Springer (2001)
4. Farach, M., Przytycka, T., Thorup, M.: On the agreement of many trees. Information Process. Lett. **55**, 297–301 (1995)
5. Guillemot, S., Berry, V.: Fixed-parameter tractability of the maximum agreement supertrees. In: Proceedings of the 18th Annual Symposium on Combinatorial Pattern Matching (CPM 2007). Lecture Notes in Computer Science. Springer, (2007)
6. Henzinger, M.R., King, V., Warnow, T.: Constructing a tree from homeomorphic subtrees, with applications to computational evolutionary biology. Algorithmica **24**(1), 1–13 (1999)
7. Hoang, V.T., Sung, W.K.: Fixed Parameter Polynomial Time Algorithms for Maximum Agreement and Compatible Supertrees. In: Albers, S., Weil, P., 25th International Symposium on Theoretical Aspects of Computer Science (STACS 2008). Dagstuhl, Germany (2007)
8. Jansson, J., Joseph, H., Ng, K., Sadakane, K., Sung, W.-K.: Rooted maximum agreement supertrees. Algorithmica **43**(4), 293–307 (2005)
9. Kao, M.-Y., Lam, T.-W., Sung, W.-K., Ting, H.-F.: An even faster and more unifying algorithm for comparing trees via unbalanced bipartite matchings. J. Algorithms **40**(2), 212–233 (2001)
10. Kearney, P.: Phylogenetics and the quartet method. In: Jiang, T., Xu, Y., Zhang, M.Q. (eds.) Current Topics in Computational Molecular Biology. The MIT Press, Massachusetts, pp. 111–133 (2002)
11. Sanderson, M.J., Purvis, A., Henze, C.: Phylogenetic supertrees: assembling the trees of life. TRENDS in Ecology & Evolution, **13**(3), 105–109 (1998)

Maximum Compatible Tree

2001; Ganapathy, Warnow

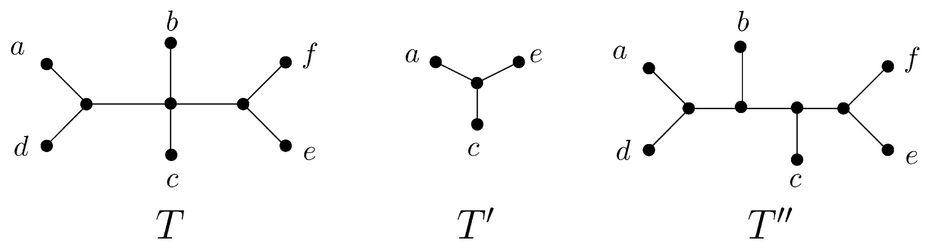
VINCENT BERRY
LIRMM, University of Montpellier, Montpellier, France

Keywords and Synonyms

Maximum refinement subtree (MRST)

Problem Definition

This problem is a pattern matching problem on leaf-labeled trees. Each input tree is considered as a branching pattern inducing specific groups of leaves. Given a tree collection with identical leaf sets, the goal is to find a largest subset of leaves on the branching pattern of which the input trees do not disagree. A *maximum compatible tree* is a tree with such a leaf-set and with the branching patterns of the input trees for these leaves. The Maximum Compatible Tree problem (MCT) is to find such a tree or, equivalently, its leaf set. The main motivation for this problem is in phylogenetics, to measure the similarity between evolutionary trees, or to represent a consensus of a set of trees. The problem was introduced in [9] and [10], under the MRST acronym. Previous related works concern the well-known Maximum Agreement Subtree prob-



Maximum Compatible Tree, Figure 1
Three unrooted trees. A tree T, a tree T' such that $T' = T\,|\{a, c, e\}$ and a tree T'' such that $T'' \trianglerighteq T$

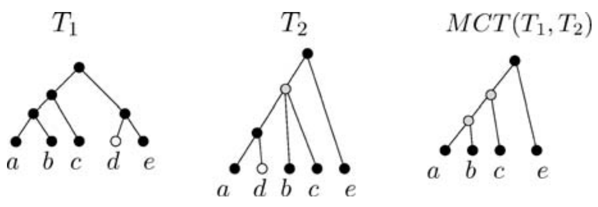
lem (MAST). Solving MAST is finding a largest subset of leaves on which all input trees *exactly* agree. More precisely, MAST seeks a tree whose branching information is isomorphic to that of a subtree in each of the input trees, while MCT seeks a tree that contains the branching information (i. e. groups) of a subtree of each input tree. This difference allows the tree obtained for MCT to be more informative, as it can include branching information present in one input tree but not in the others, as long as this information is compatible with them. Both problems are equivalent when all input trees are binary. Ganapathy and Warnow [5] were the first to give an algorithm to solve MCT in its general form. Their algorithm relies on a simple dynamic programming approach similar to a work on MAST [12] and has a running time exponential in the number of input trees and in the maximum degree of a node in the input trees. Later, [2] proposed a fixed-parameter algorithm using one parameter only. Approximation results have also been obtained [1,6], the result being low-cost polynomial-time algorithms that approximate the complement of MCT within a constant threshold.

Notations Trees considered here are evolutionary trees (*phylogenies*). Such a tree T has its leaf set $L(T)$ in bijection with a label set and is either rooted, in which case all internal nodes have at least two children each, or unrooted, in which case internal nodes have a degree of at least three. The size of $|T|$ of a tree T is the number of its leaves. Given a set L of labels and a tree T, the *restriction* of T to L, denoted $T|L$, is the tree obtained in the following way: take the smallest induced subgraph of T connecting leaves with labels in $L \cap L(T)$, then remove any degree two (non-root) node to make the tree homeomorphically irreducible. Two trees T, T' are *isomorphic*, denoted $T = T'$, if and only if there is a graph isomorphism $T \mapsto T'$ preserving leaf labels (and the root if both trees are rooted). A tree T *refines* a tree T', denoted $T \trianglerighteq T'$, whenever T can be transformed into T' by collapsing some of its internal edges (*collapsing* an edge means removing it and merging its extremities). See Fig. 1 for examples of these relations between trees.

Note that a tree T properly refining another tree T', agrees with the entire evolutionary history of T', while containing additional information absent from T': at least one high degree node of T' is replaced in T by several nodes of lesser degree, hence T contains more speciation events than T'. Given a collection $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$ of input trees with identical leaf sets L, a tree T with leaves in L is said to be *compatible with* $\mathcal{T}$ if and only if $\forall T_i \in \mathcal{T}$, $T \trianglerighteq T_i|L(T)$. If there is a tree T compatible with $\mathcal{T}$ such that $L(T) = L$, then the collection $\mathcal{T}$ is said to be *compatible*. Knowing whether a collection is compatible is a problem for which linear-time algorithms have been known for a long time e. g. [8]. The MAXIMUM COMPATIBLE TREE problem is a natural optimization version of this problem to deal with incompatible collections of trees.

Problem 1 (MAXIMUM COMPATIBLE TREE – MCT)
Input: *A collection $\mathcal{T}$ of trees with the same leaf sets.*
Output: *A tree compatible with $\mathcal{T}$ having the largest number of leaves. Such a tree is denoted* MCT($\mathcal{T}$).

See Fig. 2 for an example. Note that $\forall \mathcal{T}, |\text{MCT}(\mathcal{T})| \geq |\text{MAST}(\mathcal{T})|$ and that MCT is equivalent to MAST when input trees are binary. Note also that instances of MCT and MAST can have several optimum solutions.



Maximum Compatible Tree, Figure 2
An incompatible collection of two input trees $\{T_1, T_2\}$ and their maximum compatible tree, $T = MCT(T_1, T_2)$. Removing the leaf d renders the input trees compatible, hence $L(T) = \{a, b, c, e\}$. Here, T strictly refines T_2 restricted to $L(T)$, which is expressed by the fact that a node in T_2 (the grey one) has its child subtrees distributed between several connected nodes of T (grey nodes). Note also that here $|MCT(T_1, T_2)| > |MAST(T_1, T_2)|$

Key Results

Exact Algorithms

The MCT problem was shown to be NP-hard on 6 trees in [9], then on 2 trees in [10]. The NP-hardness holds as long as one of the input trees is not of bounded degree. For two bounded-degree trees, Hein et al. mention a polynomial-time algorithm based on *aligning* trees [10]. The work of Ganapathy and Warnow [5] proposed an exponential algorithm for solving MCT in the general case. Given two trees T_1, T_2, they show how to compute a binary MCT of any pair of subtrees ($S_1 \in T_1, S_2 \in T_2$) by dynamic programming. Subtrees whose root is of high degree are handled by considering every possible partition of the roots's children in two sets. This leads the complexity bound to have a term exponential in d, the maximum degree of a node in the input trees. When dealing with k input trees, k-tuples of subtrees are considered, and the simultaneous bipartitions of the roots's children for k subtrees are considered. Hence, the complexity bound is also exponential in k.

Theorem 1 ([5]) *Let L be a set of n leaves. The* MCT *problem for a collection of k rooted trees on L in which each tree has degree at most $d+1$, can be solved in $O(2^{2kd} n^k)$ time.*

The result easily extends to unrooted trees by considering each of the n leaves in turn as a possible root for all trees of the collection.

Theorem 2 ([5]) *Given a collection of k unrooted trees with degree at most $d+1$ on an n-leaf set, the* MCT *problem can be solved in $O(2^{2kd} n^{k+1})$.*

Let $\mathcal{T}$ be a collection on a leaf-set L, [2] considered the following decision problem, denoted MCT$_p$: given $\mathcal{T}$ and $p \in [0, n]$, does $|MCT(\mathcal{T})| \geq n - p$?

Theorem 3 ([2])
1. MCT$_p$ *on rooted trees can be solved in* $O(\min\{3^p kn, 2.27^p + kn^3\})$ *time.*
2. MCT$_p$ *on unrooted trees can be solved in* $O\big((p+1) \times \min\{3^p kn, 2.27^p + kn^3\}\big)$ *time.*

The 3^{pkn} term comes from an algorithm that first locates in $O(kn)$ time a 3-leaf set S on which the input trees conflict, then recursively obtains a maximum compatible tree T_1, resp. T_2, T_3 for each of the three collections $\mathcal{T}_1$, resp. $\mathcal{T}_2, \mathcal{T}_3$ obtained by removing from the input trees a leaf in S, and last returning the T_i such that $|T_i|$ is maximum (for $i \in [1, 3]$). The $2.27^p + kn^3$ term comes from an algorithm reducting MCT to 3-HITTING SET. Negative results have been obtained by Guillemot and Nicolas concerning the fixed-parameter tractability of MCT wrt the maximum degree D of the input trees.

Theorem 4 ([7])
1. MCT *is $W[1]$-hard with respect to D.*
2. MCT *can not be solved in $O(N^{o(2^{D/2})})$ time unless SNP $\subseteq$ SE, where N denotes the input length, i. e. $N = O(kn)$.*

The MCT problem also admits a variant that deals with *supertrees*, i. e. trees having different (but overlapping) sets of leaves. The resulting problem is $W[2]$-hard with respect to p [3].

Approximation Algorithms

The idea of locating and then eliminating successively all the conflicts between the input trees has also led to approximation algorithms for the *complement* version of the MCT problem, denoted CMCT. Let L be the leaf set of each tree in an input collection $\mathcal{T}$, CMCT aims at selecting the smallest number of leaves $S \subseteq L$ such that the collection $\{T_i|(L - S) : T_i \in \mathcal{T}\}$ is compatible.

Theorem 5 ([6]) *Given a collection $\mathcal{T}$ of k rooted trees on an n-leaf set L, there is a 3-approximation algorithm for* CMCT *that runs in $O(k^2 n^2)$ time.*

The running time of this algorithm was later improved:

Theorem 6 ([1]) *There is an $O(kn + n^2)$ time 3-approximation algorithm for* CMCT *on a collection of k rooted trees with n leaves.*

Note also that working on rooted or unrooted trees does not change the achievable approximation threshold for CMCT [1].

Applications

In bioinformatics, the MCT problem (and similarly MAST) is used to reach different practical goals. The first motivation is to measure the similarity of a set of trees. These trees can represent RNA secondary structures [10,11] or estimates of a phylogeny inferred from different datasets composed of molecular sequences (e. g. genes) [13]. The gap between the size of a maximum compatible tree and the number of input leaves indicates the degree of disimilarity of the input trees. Concerning the phylogenetic applications, quite often some edges of the trees inferred from the datasets have been collapsed due to insufficient statistical support, resulting in some higher-degree nodes in the trees considered. Each such node does not indicate a multi-speciation event but rather the uncertainty with respect to the branching pattern to be chosen for its child subtrees. In such a situation, the MCT problem is to be preferred to MAST, as it correctly handles high degree nodes, enabling them to be resolved according to branching information present in other input trees. As a result, more leaves are conserved in the output tree, hence a larger degree of similarity is detected between the input trees. Note also that a low similarity value between the input trees can be due to horizontal gene transfers. When these events are not too numerous, identifying species subject to such effects is done by first suspecting leaves discarded from a maximum compatible tree.

The shape of a maximum compatible tree, i. e. not just its size, also has an application in systematic biology to obtain a consensus of a set of phylogenies that are optimal for some tree-building criterion. For instance, the maximum parsimony and maximum likelihood criteria can provide several dozens (sometimes hundreds) of optimal or near-optimal trees. In practice, these trees are first grouped into islands of neighboring trees, and a consensus tree is obtained for each island by resorting to a classical consensus tree method, e. g. the majority-rule or strict consensus. The trees representing the islands form a collection of which a consensus is then sought. However, consensus methods keeping all input leaves tend to create trees that lack of resolution. An alternative approach lies in proposing a representative tree that contains a largest possible subset of leaves on the position of which the trees of the collection agree. Again, MCT is more suited than MAST as the input trees can contain some high-degree nodes, with the same meaning as discussed above.

Open Problems

A direction for future work is to examine the variant of MCT where some leaves are imposed in the output tree. This question arises when a biologist wants to ensure that the species central to his study are contained in the output tree. For MAST on two trees, this constrained variant of the problem was shown in a natural way to be of the same complexity as the standart version [4]. For MCT however, such a constraint can lead to several optimization problems that need to be sorted out. Another important work to be done is a set of experiments to measure the range of parameters for which the algorithms proposed to solve or approximate MCT are useful.

URL to Code

A beta-version of a Perl program can be asked to the author of this entry.

Cross References

► Maximum Agreement Subtree (of 2 Binary Trees)
► Maximum Agreement Subtree (of 3 or More Trees)

Recommended Reading

1. Berry, V., Guillemot, S., Nicolas, F., Paul, C.: On the approximation of computing evolutionary trees. In: Wang, L. (ed.) Proc. of the 11th Annual International Conference on Computing and Combinatorics (COCOON'05). LNCS, vol. 3595, pp. 115–125. Springer, Berlin (2005)
2. Berry, V., Nicolas, F.: Improved parametrized complexity of the maximum agreement subtree and maximum compatible tree problems. IEEE/ACM Trans. Comput. Biol. Bioinform. **3**(3), 289–302 (2006)
3. Berry, V., Nicolas, F.: Maximum agreement and compatible supertrees. J. Discret. Algorithms. Algorithmica, Springer, New York (2008)
4. Berry, V., Peng, Z.S., Ting, H.-F.: From constrained to unconstrained maximum agreement subtree in linear time. Algorithmica, to appear (2006)
5. Ganapathy, G., Warnow, T.J.: Finding a maximum compatible tree for a bounded number of trees with bounded degree is solvable in polynomial time. In: Gascuel, O., Moret, B.M.E. (eds.) Proc. of the 1st International Workshop on Algorithms in Bioinformatics (WABI'01), pp. 156–163 (2001)
6. Ganapathy, G., Warnow, T.J.: Approximating the complement of the maximum compatible subset of leaves of *k* trees. In: Proc. of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX'02), LCNS, vol. 2462, pp. 122–134., Springer, Berlin (2002)
7. Guillemot, S., Nicolas, F.: Solving the maximum agreement subtree and the maximum compatible tree problems on many bounded degree trees. In: Lewenshtein, M., Valiente, G. (eds.) Proc. of the 17th Combinatorial Pattern Matching Symposium (CPM'06). LNCS, vol. 4009, pp. 165–176. Springer, Berlin (2006)
8. Gusfield, D.: Efficient algorithms for inferring evolutionary trees. Networks **21**, 19–28 (1991)
9. Hamel, A.M., Steel, M.A.: Finding a maximum compatible tree is NP-hard for sequences and trees. Appl. Math. Lett. **9**(2), 55–59 (1996)
10. Hein, J., Jiang, T., Wang, L., Zhang, K.: On the complexity of comparing evolutionary trees. Discrete Appl. Math. **71**(1–3), 153–169 (1996)
11. Jiang, T., Wang, L., Zhang, K.: Alignment of trees – an alternative to tree edit. Theor. Comput. Sci. **143**(1), 137–148 (1995)
12. Steel, M.A., Warnow, T.J.: Kaikoura tree theorems: Computing the maximum agreement subtree. Inform. Process. Lett. **48**(2), 77–82 (1993)
13. Swofford, D.L., Olsen, G.J., Wadell, P.J., Hillis, D.M.: Phylogenetic inference. In: Hillis, D.M., Moritz, D.M., Mable, B.K. (eds.) Molecular systematics, 2nd edn. pp. 407–514. Sunderland, USA (1996)

Maximum-Density Segment

1994; Huang

KUN-MAO CHAO
Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan

Keywords and Synonyms

Maximum-average segment

Problem Definition

Given a sequence of numbers, $A = \langle a_1, a_2, \ldots, a_n \rangle$, and two positive integers L, U, where $1 \leq L \leq U \leq n$, the maximum-density segment problem is to find a consecutive subsequence, i. e. a segment or substring, of A with length at least L and at most U such that the average value of the numbers in the subsequence is maximized.

Key Results

If there is no length constraint, then obviously the maximum-density segment is the maximum number in the sequence. Let's first consider the problem where only the length lower bound L is imposed. By observing that the length of the shortest maximum-density segment with length at least L is at most $2L - 1$, Huang [7] gave an $O(nL)$-time algorithm. Lin et al. [10] proposed a new technique, called the *right-skew decomposition*, to partition each suffix of A into *right-skew* segments of strictly decreasing averages. The right-skew decomposition can be done in $O(n)$ time, and it can answer, for each position i, a consecutive subsequence of A starting at that position such that the average value of the numbers in the subsequence is maximized. On the basis of the right-skew decomposition, Lin et al. [10] devised an $O(n \log L)$-time algorithm for the maximum-density segment problem with a lower bound L, which was improved to $O(n)$ time by Goldwasser et al. [6]. Kim [8] gave another $O(n)$-time algorithm by reducing the problem to the maximum-slope problem in computation geometry. As for the problem which takes both L and U into consideration, Chung and Lu [4] bypassed the construction of the right-skew decomposition and gave an $O(n)$-time algorithm.

It should be noted that a closely related problem in data mining, which basically deals with a binary sequence, was independently formulated and studied by Fukuda et al. [5].

An Extension to Multiple Segments

Given a sequence of numbers, $A = \langle a_1, a_2, \ldots, a_n \rangle$, and two positive integers L and k, where $k \leq \frac{n}{L}$, let $d(A[i, j])$ denote the *density* of segment $A[i, j]$, defined as $(a_i + a_{i+1} + \cdots + a_j)/(j - i + 1)$. The problem is to find k disjoint segments $\{s_1, s_2, \ldots, s_k\}$ of A, each has a length of at least L, such that $\sum_{1 \leq i \leq k} d(s_i)$ is maximized. Chen et al. [3] proposed an $O(nkL)$-time algorithm and an improved $O(nL + k^2 L^2)$-time algorithm was given by Bergkvist and Damaschke [2]. Liu and Chao [11] gave an $O(n + k^2 L \log L)$-time algorithm.

Applications

In all organisms, the GC base composition of DNA varies between 25–75%, with the greatest variation in bacteria. Mammalian genomes typically have a GC content of 45–50%. Nekrutenko and Li [12] showed that the extent of the compositional heterogeneity in a genomic sequence strongly correlates with its GC content. Genes are found predominantly in the GC-richest isochore classes. Hence, finding GC-rich regions is an important problem in gene recognition and comparative genomics.

Given a DNA sequence, one would attempt to find segments of length at least L with the highest C+G ratio. Specifically, each of nucleotides C and G is assigned a score of 1, and each of nucleotides A and T is assigned a score of 0.

```
DNA sequence:    ATGACTCGAGCTCGTCA
Binary sequence: 00101011011011010
```

The maximum-average segments of the binary sequence correspond to those segments with the highest GC ratio in the DNA sequence. Readers can refer to [1,9,10,13,14,15] for more applications.

Open Problems

The best asymptotic time bound of the algorithms for the multiple maximum-density segments problem is $O(n + k^2 L \log L)$. Can this problem be solved in $O(n)$ time?

Cross References

► Maximum-scoring Segment with Length Restrictions

Recommended Reading

1. Arslan A., Eğecioğlu, Ö, Pevzner, P.: A new approach to sequence comparison: normalized sequence alignment. Bioinformatics **17**, 327–337 (2001)
2. Bergkvist, A., Damaschke, P.: Fast algorithms for finding disjoint subsequences with extremal densities. In: Proceedings of the 16th Annual International Symposium on Algorithms and Computation. LNCS, vol. 3827, pp. 714–723 (2005)
3. Chen, Y.H., Lu, H.I., Tang, C.Y.: Disjoint segments with maximum density. In: Proceedings of the 5th Annual International Conference on Computational Science, pp. 845–850 (2005)
4. Chung, K.-M., Lu, H.-I.: An optimal algorithm for the maximum-density segment problem. SIAM. J. Comput. **34**, 373–387 (2004)

5. Fukuda, T., Morimoto, Y., Morishita, S., Tokuyama, T.: Mining Optimized Association Rules for Numeric Attributes. J. Comput. Syst. Sci. **58**, 1–12 (1999)
6. Goldwasser, M.H., Kao, M.-Y., Lu, H.-I.: Linear-time algorithms for computing maximum-density sequence segments with bioinformatics applications. J. Comput. Syst. Sci. **70**, 128–144 (2005)
7. Huang, X.: An algorithm for identifying regions of a DNA sequence that satisfy a content requirement. Comput. Appl. Biosci. **10**, 219–225 (1994)
8. Kim, S.K.: Linear-time algorithm for finding a maximum-density segment of a sequence. Inf. Process. Lett. **86**, 339–342 (2003)
9. Lin, Y.-L., Huang, X., Jiang, T., Chao, K.-M.: MAVG: locating non-overlapping maximum average segments in a given sequence. Bioinformatics **19**, 151–152 (2003)
10. Lin, Y.-L., Jiang, T., Chao, K.-M.: Efficient algorithms for locating the length-constrained heaviest segments with applications to biomolecular sequence analysis. J. Comput. Syst. Sci. **65**, 570–586 (2002)
11. Liu, H.-F., Chao, K.-M.: On locating disjoint segments with maximum sum of densities. In: Proceedings of the 17th Annual International Symposium on Algorithms and Computation. LNCS, vol. 4288, pp. 300–307 (2006)
12. Nekrutenko, A., Li, W.H.: Assessment of compositional heterogeneity within and between eukaryotic genomes. Genome Res. **10**, 1986–1995 (2000)
13. Stojanovic, N., Florea, L., Riemer, C., Gumucio, D., Slightom, J., Goodman, M., Miller, W., Hardison, R.: Comparison of five methods for finding conserved sequences in multiple alignments of gene regulatory regions. Nucl. Acid. Res. **19**, 3899–3910 (1999)
14. Stojanovic, N., Dewar, K.: Identifying multiple alignment regions satisfying simple formulas and patterns. Bioinformatics **20**, 2140–2142 (2005)
15. Zhang, Z., Berman, P., Wiehe, T., Miller, W.: Post-processing long pairwise alignments. Bioinformatics **15**, 1012–1019 (1999)

Maximum Matching

2004; Mucha, Sankowski

MARCIN MUCHA
Faculty of Mathematics, Informatics and Mechanics, Institute of Informatics, Warsaw, Poland

Problem Definition

Let $G = (V, E)$ be an undirected graph, and let $n = |V|$, $m = |E|$. A *matching* in G is a subset $M \subseteq E$, such that no two edges of M have a common endpoint. A *perfect matching* is a matching of cardinality $n/2$. The most basic matching related problems are: finding a *maximum matching* (i. e. a matching of maximum size) and, as a special case, finding a *perfect matching* if one exists. One can also consider the case where a weight function $w\colon E \to \mathbb{R}$ is given and the problem is to find a *maximum weight matching*.

The maximum matching and maximum weight matching are two of the most fundamental algorithmic graph problems. They have also played a major role in the development of combinatorial optimization and algorithmics. An excellent account of this can be found in a classic monograph [10] by Lovász and Plummer devoted entirely to matching problems. A more up-to-date, but also more technical discussion of the subject can be found in [18].

Classical Approach

Solving the maximum matching problem in time polynomial in n is a highly non-trivial task. The first such solution was given by Edmonds [3] in 1965 and has time complexity $O(n^3)$. Edmond's ingenious algorithm uses a combinatorial approach based on augmenting paths and blossoms. Several improvements followed, culminating in the algorithm with complexity $O(m\sqrt{n})$ given by Micali and Vazirani [11] in 1980 (a complete proof of the correctness of this algorithm was given much later by Vazirani [19], a nice exposition of the algorithm and its generalization to the weighted case can be found in a work of Gabow and Tarjan [4]). Beating this bound proved very difficult, several authors managed to achieve only a logarithmic speed-up for certain values of m and n. All these algorithms essentially follow the combinatorial approach introduced by Edmonds.

The maximum matching problem is much simpler for bipartite graphs. The complexity of $O(m\sqrt{n})$ was achieved for this case already in 1971 by Hopcroft and Karp [6], while the key ideas of the first polynomial algorithms date back to 1920's and the works of König and Egerváry (see [10] and [18]).

Algebraic Approach

Around the time Micali and Vazirani introduced their matching algorithm, Lovász gave a randomized (Monte Carlo) reduction of the problem of testing whether a given n-vertex graph has a perfect matching to the problem of computing a certain determinant of a $n \times n$ matrix. Using the Hopcroft-Bunch fast Gaussian elimination algorithm [1] this determinant can be computed in time $MM(n) = O(n^\omega)$ – time required to multiply two $n \times n$ matrices. Since $\omega < 2.38$ (see [2]), for dense graphs this algorithm is asymptotically faster than the matching algorithm of Micali and Vazirani.

However, Lovász's algorithm only tests for perfect matching, it does not find it. Using it to find perfect/maximum matchings in a straightforward fashion yields algorithm with complexity $O(mn^\omega) = O(n^{4.38})$. A major open problem in the field was thus: can maximum matchings be actually found in $O(n^{\omega})$ time?

The first step in this direction was taken in 1989 by Rabin and Vazirani [15]. They showed that maximum matchings can be found in time $O(n^{\omega+1}) = O(n^{3.38})$.

Key Results

The following theorems state the key results of [12].

Theorem 1 *Maximum matching in a n-vertex graph G can be found in $O(n^3)$ time (Las Vegas) by performing Gaussian elimination on a certain matrix related to G.*

Theorem 2 *Maximum matching in an n-vertex bipartite graph can be found in $\tilde{O}(n^{\omega})$ time (Las Vegas) by performing a Hopcroft-Bunch fast Gaussian elimination on a certain matrix related to G.*

Theorem 3 *Maximum matching in an n-vertex graph can be found in $\tilde{O}(n^{\omega})$ time (Las Vegas).*

Note: $\tilde{O}$ notation suppresses polylogarithmic factors, so $\tilde{O}(f(n))$ means $O(f(n)\log^k(n))$ for some k.

Let us briefly discuss these results. Theorem 1 shows that effective matching algorithms can be simple. This is in large contrast to augmenting paths/blossoms based algorithms which a generally regarded quite complicated.

The other two theorems show that, for dense graphs, the algebraic approach is asymptotically faster than the combinatorial one.

The algorithm for the bipartite case is very simple. It's only non-elementary part is the fast matrix multiplication algorithm used as black box by the Hopcroft-Bunch algorithm. The general algorithm, however, is complicated and uses strong structural results from matching theory. A natural question is whether or not it is possible to give a simpler and/or purely algebraic algorithm. This has been positively answered by Harvey [5].

Several other related results followed. Mucha and Sankowski [13] showed that maximum matchings in planar graphs can be found in time $\tilde{O}(n^{\omega/2}) = \tilde{O}(n^{1.19})$ which is currently fastest known. Yuster and Zwick [20] extended this to any excluded minor class of graphs. Sankowski [16] gave an RNC work-efficient matching algorithm (see also Mulmuley et al. [14] and Karp et al. [8] for earlier, less efficient RNC matching algorithms, and Karloff [7] for a description of a general technique for making such algorithm Las Vegas). He also generalized Theorem 2 to the case of weighted bipartite graphs with integer weights from $[0, \ldots, W]$, showing that in this case maximum weight matchings can be found in time $\tilde{O}(Wn^{\omega})$ (see [17]).

Applications

The maximum matching problem has numerous applications, both in practice and as a subroutine in other algorithms. A nice discussion of practical applications can be found in the monograph [10] by Lovász and Plummer. It should be noted, however, that algorithms based on fast matrix multiplication are completely impractical, so the results discussed here are not really useful in these applications.

On the theoretical side, faster maximum (weight) matching algorithms yield faster algorithms for related problems: disjoint s-t paths problem, the minimum (weight) edge cover problem, the (maximum weight) b-matching problem, the (maximum weight) b-factor problem, the maximum (weight) T-join or the Chinese postman problem. For detailed discussion of all these applications see [10] and [18].

The algebraic algorithm of Theorem 1 also has a significant educational value. The combinatorial algorithms for the general maximum matching problem are generally regarded too complicated for an undergraduate course. That is definitely not the case with the algebraic $O(n^3)$ algorithm.

Open Problems

One of the most important open problems in the area is generalizing the results discussed above to weighted graphs. Sankowski [17] gives a $\tilde{O}(Wn^{\omega})$ algorithm for bipartite graphs with integer weights from the interval $[0..W]$. The complexity of this algorithm is really bad in terms of W. No effective algebraic algorithm is known for general weighted graphs.

Another interesting, but most likely very hard problem is the derandomization of the algorithms discussed.

Cross References

► All Pairs Shortest Paths via Matrix Multiplication
► Assignment Problem

Recommended Reading

1. Bunch, J., Hopcroft, J.: Triangular Factorization and Inversion by Fast Matrix Multiplication. Math. Comput. **125**, 231–236 (1974)
2. Coppersmith, D., Winograd, S.: Matrix Multiplication via Arithmetic Progressions. In: Proceedings of the 19th Annual ACM Conference on Theory of Computing (STOC), 1987, pp. 1–6
3. Edmonds, J.: Paths, Trees, and Flowers. Canad. J. Math. **17**, 449–467 (1965)
4. Gabow, H.N., Tarjan, R.E.: Faster scaling algorithms for general graph matching problems. J. ACM **38**(4), 815–853 (1991)

5. Harvey, N.: Algebraic Structures and Algorithms for Matching and Matroid Problems. In: Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS), 2006
6. Hopcroft, J.E., Karp, R.M.: An $O(n^{5/2})$ Algorithm for Maximum Matchings in Bipartite Graphs. SIAM J. Comput. **2**, 225–231 (1973)
7. Karloff, H.: A Las Vegas RNC algorithm for maximum matching. Combinatorica **6**, 387–391 (1986)
8. Karp, R., Upfal, E., Widgerson, A.: Constructing a perfect matching is in Random NC. Combinatorica **6**, 35–48 (1986)
9. Lovász, L.: On Determinants, Matchings and Random Algorithms. In: Budach, L. (ed.) Fundamentals of Computation Theory, FCT'79, pp. 565–574. Akademie-Verlag, Berlin (1979)
10. Lovász, L., Plummer, M.D.: Matching Theory. Akadémiai Kiadó – North Holland, Budapest (1986)
11. Micali, S., Vazirani, V.V.: An $O(\sqrt{V}E)$ Algorithm for Finding Maximum Matching in General Graphs. In: Proceedings of the 21st Annual IEEE Symposium on Foundations of Computer Science (FOCS), 1980, pp. 17–27
12. Mucha, M., Sankowski, P.: Maximum Matchings via Gaussian Elimination. In: Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS), 2004 pp. 248–255
13. Mucha, M., Sankowski, P.: Maximum Matchings in Planar Graphs via Gaussian Elimination. Algorithmica **45**, 3–20 (2006)
14. Mulmuley, K., Vazirani, U.V., Vazirani, V.V.: Matching is as easy as matrix inversion. In: Proceedings of the 19th Annual ACM Conference on Theory of Computing, pp. 345–354. ACM Press, New York (1987)
15. Rabin, M.O., Vazirani, V.V.: Maximum Matchings in General Graphs Through Randomization. J. Algorithms **10**, 557–567 (1989)
16. Sankowski, P.: Processor Efficient Parallel Matching. In: Proceeding of the 17th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), 2005, pp. 165–170
17. Sankowski, P.: Weighted Bipartite Matching in Matrix Multiplication Time. In: Proceedings of the 33rd International Colloquium on Automata, Languages and Programming, 2006, pp. 274–285
18. Schrijver, A.: Combinatorial optimization: polyhedra and efficiency. Springer, Berlin Heidelberg (2003)
19. Vazirani, V.V.: A Theory of Alternating Paths and Blossoms for Proving Correctness of the $O(\sqrt{V}E)$ Maximum Matching Algorithm. Combinatorica **14**(1), 71–109 (1994)
20. Yuster, R., Zwick, U.: Maximum Matching in Graphs with an Excluded Minor. In: Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA), 2007

Maximum-scoring Segment with Length Restrictions

2002; Lin, Jiang, Chao

Kun-Mao Chao
Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan

Keywords and Synonyms

Shortest path; Longest path

Problem Definition

Given a sequence of numbers, $A = \langle a_1, a_2, \dots, a_n \rangle$, and two positive integers L, U, where $1 \le L \le U \le n$, the maximum-sum segment problem is to find a consecutive subsequence, i. e. a segment or substring, of A with length at least L and at most U such that the sum of the numbers in the subsequence is maximized.

Key Results

The maximum-sum segment problem without length constraints is linear-time solvable by using Kadane's algorithm [2]. Huang extended the recurrence relation used in [2] for solving the maximum-sum segment problem, and derived a linear-time algorithm for computing the maximum-sum segment with length at least L. Lin et al. [10] proposed an $O(n)$-time algorithm for the maximum-sum segment problem with both L and U constraints, and an online version was given by Fan et al. [8].

An Extension to Multiple Segments

Computing the k largest sums over all possible segments is a natural extension of the maximum-sum segment problem. This extension has been considered from two perspectives, one of which allows the segments to overlap, while the other disallows.

Linear-time algorithms for finding all the non-overlapping maximal segments were given in [3,12]. On the other hand, one may focus on finding the k maximum-sum segments whose overlapping is allowed. A naïve approach is to choose the k largest from the sums of all possible contiguous subsequences which requires $O(n^2)$ time. Bae and Takaoka [1] presented an $O(kn)$-time algorithm for the k maximum segment problem. Liu and Chao [11] noted that the k maximum-sum segments problem can be solved in $O(n + k)$ time [7], and gave an $O(n + k)$-time algorithm for the Length-Constrained k Maximum-Sum Segments Problem.

Applications

The algorithms for the maximum-sum segment problem have applications in finding GC-rich regions in a genomic DNA sequence, postprocessing sequence alignments, and annotating multiple sequence alignments. Readers can refer to [3,4,5,6,10,12,13,14,15] for more details.

Open Problems

It would be interesting to consider the higher dimensional cases.

Cross References

▶ Maximum-Density Segment

Recommended Reading

1. Bae, S.E., Takaoka, T.: Algorithms for the problem of k maximum sums and a VLSI algorithm for the k maximum subarrays problem. Proceedings of the 7th International Symposium on Parallel Architectures, Algorithms and Networks, pp. 247–253 (2004)
2. Bentley, J.: Programming Pearls. Addison-Wesley, Reading (1986)
3. Chen, K.-Y., Chao, K.-M.: On the range maximum-sum segment query problem. Proceedings of the 15th International Symposium on Algorithms And Computation. LNCS **3341**, 294–305 (2004)
4. Chen, K.-Y., Chao, K.-M.: Optimal algorithms for locating the longest and shortest segments satisfying a sum or an average constraint. Inf. Process. Lett. **96**, 197–201 (2005)
5. Cheng, C.-H., Chen, K.-Y., Tien, W.-C., Chao, K.-M.: Improved algorithms for the *k* maximum-sum problems. Proceedings of the 16th International Symposium on Algorithms And Computation. Theoret. Comput. Sci. 362: 162–170 (2006)
6. Csűrös, M.: Maximum-scoring segment sets. IEEE/ACM Trans. Comput. Biol. Bioinform. **1**, 139–150 (2004)
7. Eppstein, D.: Finding the *k* Shortest Paths. SIAM J. Comput. **28**, 652–673 (1998)
8. Fan, T.-H., Lee, S., Lu, H.-I., Tsou, T.-S., Wang, T.-C., Yao, A.: An optimal algorithm for maximum-sum segment and its application in bioinformatics. Proceedings of the Eighth International Conference on Implementation and Application of Automata. LNCS **2759**, 251–257 (2003)
9. Huang, X.: An algorithm for identifying regions of a DNA sequence that satisfy a content requirement. Comput. Appl. Biosci. **10**, 219–225 (1994)
10. Lin, Y.-L., Jiang, T., Chao, K.-M.: Efficient algorithms for locating the length-constrained heaviest segments with applications to biomolecular sequence analysis. J. Comput. Syst. Sci. **65**, 570–586 (2002)
11. Liu, H.-F., Chao, K.-M.: Algorithms for Finding the Weight-Constrained *k* Longest Paths in a Tree and the Length-Constrained *k* Maximum-Sum Segments of a Sequence. Theoret. Comput. Sci. in revision (2008)
12. Ruzzo, W.L., Tompa, M.: A linear time algorithm for finding all maximal scoring subsequences. Proceedings of the 7th International Conference on Intelligent Systems for Molecular Biology, pp. 234–241 (1999)
13. Stojanovic, N., Florea, L., Riemer, C., Gumucio, D., Slightom, J., Goodman, M., Miller, W., Hardison, R.: Comparison of five methods for finding conserved sequences in multiple alignments of gene regulatory regions. Nucleic Acids Res. **19**, 3899–3910 (1999)
14. Stojanovic, N., Dewar, K.: Identifying multiple alignment regions satisfying simple formulas and patterns. Bioinformatics **20**, 2140–2142 (2005)
15. Zhang, Z., Berman, P., Wiehe, T., Miller, W.: Post-processing long pairwise alignments. Bioinformatics **15**, 1012–1019 (1999)

Maximum Two-Satisfiability

2004; Williams

RYAN WILLIAMS
Department of Computer Science,
Carnegie Mellon University, Pittsburgh, PA, USA

Keywords and Synonyms

Max 2-SAT

Problem Definition

In the maximum 2-satisfiability problem (abbreviated as MAX 2-SAT), one is given a Boolean formula in conjunctive normal form, such that each clause contains at most two literals. The task is to find an assignment to the variables of the formula such that a maximum number of clauses is satisfied.

MAX 2-SAT is a classic optimization problem. Its decision version was proved NP-complete by Garey, Johnson, and Stockmeyer [7], in stark contrast with 2-SAT which is solvable in linear time [2]. To get a feeling for the difficulty of the problem, the NP-completeness reduction is sketched here. One can transform any 3-SAT instance F into a MAX 2-SAT instance F', by replacing each clause of F such as

$$c_i = (\ell_1 \vee \ell_2 \vee \ell_3) ,$$

where ℓ_1, ℓ_2, and ℓ_3 are arbitrary literals, with the collection of 2-CNF clauses

$$(\ell_1), (\ell_2), (\ell_3), (c_i), (\neg\ell_1 \vee \neg\ell_2), (\neg\ell_2 \vee \neg\ell_3), \\ (\neg\ell_1 \vee \neg\ell_3), (\ell_1 \vee c_i), (\ell_2 \vee c_i), (\ell_3 \vee c_i) ,$$

where c_i is a new variable. The following are true:

- If an assignment satisfies c_i, then exactly seven of the ten clauses in the 2-CNF collection can be satisfied.
- If an assignment does not satisfy c_i, then exactly six of the ten clauses can be satisfied.

If F is satisfiable then there is an assignment satisfying 7/10 of the clauses in F', and if F is not satisfiable then no assignment satisfies more than 7/10 of the clauses in F'. Since 3-SAT reduces to MAX 2-SAT, it follows that MAX 2-SAT (as a decision problem) is NP-complete.

Notation

A CNF formula is represented as a set of clauses.

The symbols $\mathbb{R}$ and $\mathbb{Z}$ denote the sets of reals and integers, respectively. The letter ω denotes the smallest real number such that for all $\epsilon > 0$, n by n matrix multiplication over a ring can be performed in $O(n^{\omega+\epsilon})$ ring operations. Currently, it is known that $\omega < 2.376$ [4]. The ring matrix product of two matrices A and B is denoted by $A \times B$.

Let A and B be matrices with entries from $\mathbb{R} \cup \{\infty\}$. The *distance product* of A and B (written shorthand as $A \otimes_d B$) is the matrix C defined by the formula

$$C[i, j] = \min_{k=1,\ldots,n} \{A[i, k] + B[k, j]\} .$$

A word on m's and n's: in reference to graphs, m and n denote the number of edges and the number of nodes in the graph, respectively. In reference to CNF formulas, m and n denote the number of clauses and the number of variables, respectively.

Key Result

The primary result of this article is a procedure solving MAX 2-SAT in $O(m \cdot 2^{\omega n/3})$ time. The method can be generalized to *count* the number of solutions to *any* constraint optimization problem with at most two variables per constraint (cf. [17]), though the presentation in this article shall be somewhat different from the reference, and much simpler. There are several other known exact algorithms for MAX 2-SAT that are more effective in special cases, such as sparse instances [3,8,9,11,12,13,15,16]. The procedure described below is the only one known (to date) that runs in $O(poly(m) \cdot 2^{\delta n})$ time (for some fixed $\delta < 1$) in all possible cases.

Key Idea

The algorithm gives a reduction from MAX 2-SAT to the problem MAX TRIANGLE, in which one is given a graph with integer weights on its nodes and edges, and the goal is to output a 3-cycle of maximum weight. At first, the existence of such a reduction sounds strange, as MAX TRIANGLE can be trivially solved in $O(n^3)$ time by trying all possible 3-cycles. The key is that the reduction exponentially increases the problem size, from a MAX 2-SAT instance with m clauses and n variables, to a MAX TRIANGLE instance having $O(2^{2n/3})$ edges, $O(2^{n/3})$ nodes, and weights in the range $\{-m, \ldots, m\}$.

Note that if MAX TRIANGLE required $\Theta(n^3)$ time to solve, then the resulting MAX 2-SAT algorithm would take $\Theta(2^n)$ time, rendering the above reduction pointless. However, it turns out that the brute-force search of $O(n^3)$ for MAX TRIANGLE is not the best one can do– using fast matrix multiplication, there is an algorithm for MAX TRIANGLE that runs in $O(Wn^{\omega})$ time on graphs with weights in the range $\{-W, \ldots, W\}$.

Main Algorithm

First, a reduction from MAX 2-SAT to MAX TRIANGLE is described, arguing that each triangle of weight K in the resulting graph is in one-to-one correspondence with an assignment that satisfies K clauses of the MAX 2-SAT instance. Let a, b be reals, and let $\mathbb{Z}[a, b] := [a, b] \cap \mathbb{Z}$

Lemma 1 *If* MAX TRIANGLE *on graphs with n nodes and weights in* $\mathbb{Z}[-W, W]$ *is solvable in* $O(f(W) \cdot g(n))$ *time, for polynomials f and g, then* MAX 2-SAT *is solvable in* $O(f(m) \cdot g(2^{n/3}))$ *time, where m is the number of clauses and n is the number of variables.*

Proof Let C be a given 2-CNF formula. Assume without loss of generality that n is divisible by 3. Let F be an instance of MAX 2-SAT. Arbitrarily partition the n variables of F into three sets P_1, P_2, P_3, each having $n/3$ variables. For each P_i, make a list L_i of all $2^{n/3}$ assignments to the variables of P_i.

Define a graph $G = (V, E)$ with $V = L_1 \cup L_2 \cup L_3$ and $E = \{(u, v) | u \in P_i, v \in P_j, i \neq j\}$. That is, G is a complete tripartite graph with $2^{n/3}$ nodes in each part, and each node in G corresponds to an assignment to $n/3$ variables in C. Weights are placed on the nodes and edges of G as follows. For a node v, define $w(v)$ to be the number of clauses that are satisfied by the partial assignment denoted by v. For each edge $\{u, v\}$, define $w(\{u, v\}) = -W_{uv}$, where W_{uv} is the number of clauses that are satisfied by *both* u and v.

Define the weight of a triangle in G to be the total sum of all weights and nodes in the triangle.

Claim 1 There is a one-to-one correspondence between the triangles of weight K in G and the variable assignments satisfying exactly K clauses in F.

Proof Let a be a variable assignment. Then there exist unique nodes $v_1 \in L_1, v_2 \in L_2$, and $v_3 \in L_3$ such that a is precisely the concatenation of v_1, v_2, v_3 as assignments. Moreover, any triple of nodes $v_1 \in L_1, v_2 \in L_2$, and $v_3 \in L_3$ corresponds to an assignment. Thus there is a one-to-one correspondence between triangles in G and assignments to F.

The number of clauses satisfied by an assignment is exactly the weight of its corresponding triangle. To see this, let $T_a = \{v_1, v_2, v_3\}$ be the triangle in G corresponding to assignment a. Then

$$\begin{aligned} w(T_a) &= w(v_1) + w(v_2) + w(v_3) + w(\{v_1, v_2\}) \\ &\quad + w(\{v_2, v_3\}) + w(\{v_1, v_3\}) \\ &= \sum_{i=1}^{3} |\{c \in F | v_i \text{ satisfies } F\}| \\ &\quad - \sum_{i,j:i \neq j} |\{c \in F | v_i \text{ and } v_j \text{ satisfy } F\}| \\ &= |\{c \in F | a \text{ satisfies } F\}| , \end{aligned}$$

where the last equality follows from the inclusion-exclusion principle. □

Notice that the number of nodes in G is $3 \cdot 2^{n/3}$, and the absolute value of any node and edge weight is m. Therefore, running a MAX TRIANGLE algorithm on G, a solution to MAX 2-SAT is obtained in $O(f(m) \cdot g(3 \cdot 2^{n/3}))$, which is $O(f(m) \cdot g(2^{n/3}))$ since g is a polynomial. This completes the proof of Lemma 1. □

Next, a procedure is described for finding a maximum triangle faster than brute-force search, using fast matrix multiplication. Alon, Galil, and Margalit [1] (following Yuval [20]) showed that the distance product for matrices with entries drawn from $\mathbb{Z}[-W, W]$ can be computed using fast matrix multiplication as a subroutine.

Theorem 2 (Alon, Galil, Margalit [1]) *Let A and B be $n \times n$ matrices with entries from $\mathbb{Z}[-W, W] \cup \{\infty\}$. Then $A \otimes_d B$ can be computed in $O(Wn^{\omega} \log n)$ time.*

Proof (Sketch) One can replace ∞ entries in A and B with $2W + 1$ in the following. Define matrices A' and B', where

$$A'[i, j] = x^{3W - A[i,j]} , \quad B'[i, j] = x^{3W - B[i,j]} ,$$

and x is a variable. Let $C = A' \times B'$. Then

$$C[i, j] = \sum_{k=1}^{n} x^{6W - A[i,k] - B[k,j]} .$$

The next step is to pick a number x that makes it easy to determine, from the sum of arbitrary powers of x, the largest power of x appearing in the sum; this largest power immediately gives the minimum $A[i, k] + B[k, j]$. Each $C[i,j]$ is a polynomial in x with coefficients from $\mathbb{Z}[0, n]$. Suppose each $C[i,j]$ is evaluated at $x = (n + 1)$. Then each entry of $C[i,j]$ can be seen as an $(n + 1)$-ary number, and the position of this number's most significant digit gives the minimum $A[i, k] + B[k, j]$.

In summary, $A \otimes_d B$ can be computed by constructing

$$A'[i, j] = (n+1)^{3W - A[i,j]} , \quad B'[i, j] = (n+1)^{3W - B[i,j]}$$

in $O(W \log n)$ time per entry, computing $C = A' \times B'$ in $O(n^{\omega} \cdot (W \log n))$ time (as the sizes of the entries are $O(W \log n)$), then extracting the minimum from each entry of C, in $O(n^2 \cdot W \log n)$ time. Note if the minimum for an entry $C[i,j]$ is at least $2W + 1$, then $C[i, j] = \infty$. □

Using the fast distance product algorithm, one can solve MAX TRIANGLE faster than brute-force. The following is based on an algorithm by Itai and Rodeh [10] for detecting if an unweighted graph has a triangle in less than n^3 steps. The result can be generalized to *counting* the number of k-*cliques*, for arbitrary $k \geq 3$. (To keep the presentation simple, the counting result is omitted. Concerning the k-clique result, there is unfortunately no asymptotic runtime benefit from using a k-clique algorithm instead of a triangle algorithm, given the current best algorithms for these problems.)

Theorem 3 MAX TRIANGLE *can be solved in $O(W n^{\omega} \log n)$, for graphs with weights drawn from $\mathbb{Z}[-W, W]$.*

Proof First, it is shown that a weight function on nodes and edges can be converted into an equivalent weight function with weights on only edges. Let w be the weight function of G, and redefine the weights to be:

$$w'(\{u, v\}) = \frac{w(u) + w(v)}{2} + w(\{u, v\}) , \quad w'(u) = 0 .$$

Note the weight of a triangle is unchanged by this reduction.

The next step is to use a fast distance product to find a maximum weight triangle in an edge-weighted graph of n nodes. Construe the vertex set of G as the set $\{1, \ldots, n\}$. Define A to be the $n \times n$ matrix such that $A[i, j] = -w(\{i, j\})$ if there is an edge $\{i, j\}$, and $A[i, j] = \infty$ otherwise. The claim is that there is a triangle through node i of weight at least K if and only if $(A \otimes_d A \otimes_d A)[i, i] \leq -K$. This is because $(A \otimes_d A \otimes_d A)[i, i] \leq -K$ if and only if there are distinct j and k such that $\{i, j\}, \{j, k\}, \{k, i\}$ are edges and $A[i, j] + A[j, k] + A[k, i] \leq -K$, i. e., $w(\{i, j\}) + w(\{j, k\}) + w(\{k, i\}) \geq K$.

Therefore, by finding an i such that $(A \otimes_d A \otimes_d A)[i, i]$ is minimized, one obtains a node i contained in a maximum triangle. To obtain the actual triangle, check all m edges $\{j, k\}$ to see if $\{i, j, k\}$ is a triangle. □

Theorem 4 MAX 2-SAT *can be solved in $O(m \cdot 1.732^n)$ time.*

Proof Given a set of clauses C, apply the reduction from Lemma 1 to get a graph G with $O(2^{n/3})$ nodes and weights from $\mathbb{Z}[-m, m]$. Apply the algorithm of Theorem 3 to output a max triangle in G in $O(m \cdot 2^{\omega n/3} \log(2^{n/3})) =$

$O(m \cdot 1.732^n)$ time, using the $O(n^{2.376})$ matrix multiplication of Coppersmith and Winograd [4]. □

Applications

By modifying the graph construction, one can solve other problems in $O(1.732^n)$ time, such as MAX CUT, MINIMUM BISECTION, and SPARSEST CUT. In general, any constraint optimization problem for which each constraint has at most two variables can be solved faster using the above approach. For more details, see [17] and the recent survey by Woeginger [19]. Techniques similar to the above algorithm have also been used by Dorn [6] to speed up dynamic programming for some problems on planar graphs (and in general, graphs of bounded branch-width).

Open Problems

- Improve the space usage of the above algorithm. Currently, $\Theta(2^{2n/3})$ space is needed. A very interesting open question is if there is a $O(1.99^n)$ time algorithm for MAX 2-SAT that uses only *polynomial* space. This question would have a positive answer if one could find an algorithm for solving the k-CLIQUE problem that uses polylogarithmic space and $n^{k-\delta}$ time for some $\delta > 0$ and $k \geq 3$.
- Find a faster-than-2^n algorithm for MAX 2-SAT that does not require fast matrix multiplication. The fast matrix multiplication algorithms have the unfortunate reputation of being impractical.
- Generalize the above algorithm to work for MAX k-SAT, where k is any positive integer. The current formulation would require one to give an efficient algorithm for finding a small hyperclique in a hypergraph. However, no general results are known for this problem. It is conjectured that for all $k \geq 2$, MAX k-SAT is in $\tilde{O}(2^{n(1-\frac{1}{k+1})})$ time, based on the conjecture that matrix multiplication is in $n^{2+o(1)}$ time [17].

Cross References

► All Pairs Shortest Paths via Matrix Multiplication
► Max Cut
► Minimum Bisection
► Sparsest Cut

Recommended Reading

1. Alon, N., Galil, Z., Margalit, O.: On the exponent of the all-pairs shortest path problem. J. Comput. Syst. Sci. **54**, 255–262 (1997)
2. Aspvall, B., Plass, M.F., Tarjan R.E.: A linear-time algorithm for testing the truth of certain quantified boolean formulas. Inf. Proc. Lett. **8**(3), 121–123 (1979)
3. Bansal, N., Raman, V.: Upper bounds for Max Sat: Further Improved. In: Proceedings of ISAAC. LNCS, vol. 1741, pp. 247–258. Springer, Berlin (1999)
4. Coppersmith, D., Winograd S.: Matrix Multiplication via Arithmetic Progressions. JSC **9**(3), 251–280 (1990)
5. Dantsin, E., Wolpert, A.: Max SAT for formulas with constant clause density can be solved faster than in $O(2^n)$ time. In: Proc. of the 9th International Conference on Theory and Applications of Satisfiability Testing. LNCS, vol. 4121, pp. 266–276. Springer, Berlin (2006)
6. Dorn, F.: Dynamic Programming and Fast Matrix Multiplication. In: Proceedings of 14th Annual European Symposium on Algorithms. LNCS, vol. 4168, pp. 280–291. Springer, Berlin (2006)
7. Garey, M., Johnson, D., Stockmeyer, L.: Some simplified NP-complete graph problems. Theor. Comput. Sci. **1**, 237–267 (1976)
8. Gramm, J., Niedermeier, R.: Faster exact solutions for Max2Sat. In: Proceedings of CIAC. LNCS, vol. 1767, pp. 174–186. Springer, Berlin (2000)
9. Hirsch, E.A.: A $2^{m/4}$-time Algorithm for Max 2-SAT: Corrected Version. Electronic Colloquium on Computational Complexity Report TR99-036 (2000)
10. Itai, A., Rodeh, M.: Finding a Minimum Circuit in a Graph. SIAM J. Comput. **7**(4), 413–423 (1978)
11. Kneis, J., Mölle, D., Richter, S., Rossmanith, P.: Algorithms Based on the Treewidth of Sparse Graphs. In: Proc. Workshop on Graph Theoretic Concepts in Computer Science. LNCS, vol. 3787, pp. 385–396. Springer, Berlin (2005)
12. Kojevnikov, A., Kulikov, A.S.: A New Approach to Proving Upper Bounds for Max 2-SAT. In: Proc. of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 11–17 (2006)
13. Mahajan, M., Raman, V.: Parameterizing above Guaranteed Values: MAXSAT and MAXCUT. J. Algorithms **31**(2), 335–354 (1999)
14. Niedermeier, R., Rossmanith, P.: New upper bounds for maximum satisfiability. J. Algorithms **26**, 63–88 (2000)
15. Scott, A., Sorkin, G.: Faster Algorithms for MAX CUT and MAX CSP, with Polynomial Expected Time for Sparse Instances. In: Proceedings of RANDOM-APPROX 2003. LNCS, vol. 2764, pp. 382–395. Springer, Berlin (2003)
16. Williams, R.: On Computing k-CNF Formula Properties. In: Theory and Applications of Satisfiability Testing. LNCS, vol. 2919, pp. 330–340. Springer, Berlin (2004)
17. Williams, R.: A new algorithm for optimal 2-constraint satisfaction and its implications. Theor. Comput. Sci. **348**(2–3), 357–365 (2005)
18. Woeginger, G.J.: Exact algorithms for NP-hard problems: A survey. In: Combinatorial Optimization – Eureka! You shrink! LNCS, vol. 2570, pp. 185–207. Springer, Berlin (2003)
19. Woeginger, G.J.: Space and time complexity of exact algorithms: some open problems. In: Proc. 1st Int. Workshop on Parameterized and Exact Computation (IWPEC 2004). LNCS, vol. 3162, pp. 281–290. Springer, Berlin (2004)
20. Yuval, G.: An Algorithm for Finding All Shortest Paths Using $N^{2.81}$ Infinite-Precision Multiplications. Inf. Process. Lett. **4**(6), 155–156 (1976)

Max Leaf Spanning Tree

2005; Estivill-Castro, Fellows, Langston, Rosamond

FRANCES ROSAMOND
Parameterized Complexity Research Unit,
University of Newcastle, Callaghan, NSW, Australia

Keywords and Synonyms

Maximum leaf spanning tree; Connected dominating set; Extremal structure

Problem Definition

The MAX LEAF SPANNING TREE problem asks us to find a spanning tree with at least k leaves in an undirected graph. The decision version of parameterized MAX LEAF SPANNING TREE is the following:

MAX LEAF SPANNING TREE
INPUT: A connected graph G, and an integer k.
PARAMETER: An integer k.
QUESTION: Does G have a spanning tree with at least k leaves?

The parameterized complexity of the nondeterministic polynomial-time complete MAX LEAF SPANNING TREE problem has been extensively studied [2,3,9,11] using a variety of kernelization, branching and other fixed-parameter tractable (FPT) techniques. The authors are the first to propose an extremal structure method for hard computational problems. The method, following in the sense of Grothendieck and in the spirit of the graph minors project of Robertson and Seymour, is that a mathematical project should unfold as a series of small steps in an overall trajectory that is described by the appropriate "mathematical machine." The authors are interested in statements of the type: Every connected graph on n vertices that satisfies a certain set of properties has a spanning tree with at least k leaves, and this spanning tree can be found in time $O(f(k) + n^c)$, where c is a constant (independent of k) and f is an arbitrary function.

In parameterized complexity, the value k is called the *parameter* and is used to capture some structure of the input or other aspect of the computational objective. For example, k might be the number of edges to be deleted in order to obtain a graph with no cycles, or k might be the number of DNA sequences to be aligned in an alignment, or k may be the maximum type-declaration nesting depth of a compiler, or $k = 1/\epsilon$ may be the parameterization in the analysis of approximation, or k might be a composite of several variables.

There are two important ways of comparing FPT algorithms, giving rise to two FPT *races*. In the "$f(k)$" race, the competition is to find ever more slowing growing parameter functions $f(k)$ governing the complexity of FPT algorithms. The "kernelization race" refers to the following lemma stating that a problem is in FPT if and only if the input can be preprocessed (*kernelized*) in "ordinary" polynomial time into an instance whose size is bounded by a function of k only.

Lemma 1 *A parameterized problem Π is in FPT if and only if there is a polynomial-time transformation (in both n and k) that takes (x, k) to (x', k') such that:*
(1) (x, k) is a yes-instance of Π if and only if (x', k') is a yes-instance of Π,
(2) $k' \leq k$, and
(3) $|x'| \leq g(k)$ for some fixed function g.

In the situation described by the lemma, say that we can *kernelize* to instances of size at most $g(k)$. Although the two races are often closely related, the result is not always the same. The current best FPT algorithm for MAX LEAF is due to Bonsma [1] (following the extremal structure approach outlined by the authors) with a running time of $O^*(8.12^k)$ to determine whether a graph G on n vertices has a spanning tree with at least k leaves; however the authors present the FPT algorithm with the smallest kernel size.

The authors list five independent deliverables associated to the extremal structure theory, and illustrate all of the objectives for the MAX LEAF problem. The five objectives are:

(A) Better FPT algorithms as a result of deeper structure theory, more powerful reduction rules associated with that structure theory, and stronger inductive proofs of improved kernelization bounds.
(B) Powerful preprocessing (*data reduction/kernelization*) rules and *combinations* of rules that can be used regardless of whether the parameter is small and that can be combined with other approaches, such as approximation and heuristics. These are usually easy to program.
(C) Gradients and transformation rules for local search heuristics.
(D) Polynomial-time approximation algorithms and performance bounds proved in a systematic way.
(E) Structure to exploit for solving other problems.

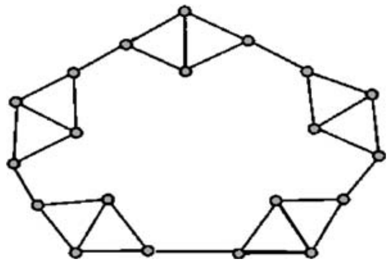
Key Results

The key results are programmatic, providing a *method of extremal structure* as a systematic method for designing FPT algorithms. The five interrelated objectives listed above are surveyed, and each is illustrated using the MAX LEAF SPANNING TREE problem.

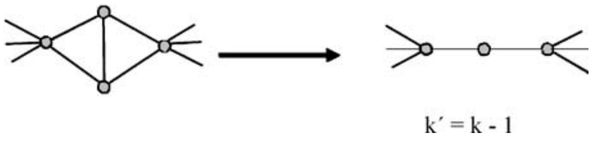
Objective A: FPT Algorithms

The objective here is to find polynomial-time preprocessing (*kernelization*) rules where $g(k)$ is as small as possible. This has a direct payoff in terms of program objective B.

Rephrased as a structure theory question, the crucial issue is: *What is the structure of graphs that do not have a subgraph with k leaves?* A graph theory result due to Kleitman and West shows that a graph of minimum degree at least 3, that excludes a k-leaf subgraph, has at most $4(k-3)$ vertices. Figure 1 shows that this is the best possible result for this hypothesis. However, investigating the structure using extremal methods reveals the need for the reduction rule of Fig. 2. About 20 different polynomial-time reduction rules (some much more complex and "global" in structure than the simple local reduction rule depicted) are sufficient to kernelize to a graph of minimum degree 2 having at most $3.5k$ vertices.



Max Leaf Spanning Tree, Figure 1
Reduction rules were developed in order to reduce this Kleitman–West graph structure



Max Leaf Spanning Tree, Figure 2
A reduction rule for the Kleitman–West graph

In general, an instance of a parameterized problem consists of a pair (x, k) and a "boundary" which is located by holding x fixed and varying k and regarding whether the outcome of the decision problem is *yes* or *no*. Of interest is the boundary when x is reduced. A typical boundary lemma looks like the following.

Lemma 2 *Suppose (G, k) is a reduced instance of* MAX LEAF, *with (G, k) a yes-instance and $(G, k+1)$ a no-instance. Then $|G| \leq ck$. (Here c is a small constant that becomes clarified during the investigation.)*

A proof of a boundary lemma is by minimum counterexample. A counterexample would be a graph such that (1) (G, k) is reduced, (2) (G, k) is a *yes*-instance of MAX LEAF, (3) $(G, k+1)$ is a *no*-instance, and (4) $|G| > ck$.

The proof of a boundary lemma unfolds gradually. Initially, it is not known what bound will eventually succeed and it is not known exactly what is meant by *reduced*. In the course of an attempted proof, these details are worked out. As the arguments unfold, structural situations will suggest new reduction rules. Strategic choices involved in a boundary lemma include:

(1) Determining the polarity of the boundary, and setting up the boundary lemma.
(2) Choosing a witness structure.
(3) Setting inductive priorities.
(4) Developing a series of structural claims that describe the situation at the boundary.
(5) Discovering reduction rules that can act in polynomial-time on relevant structural situations at the boundary.
(6) As the structure at the boundary becomes clear, filling in the blank regarding the kernelization bound.

The overall structure of the argument is "by minimum counterexample" according to the priorities established by choice 3, which generally make reference to choice 2. The proof proceeds by a series of small steps consisting of structural claims that lead to a detailed structural picture at the "boundary"—and thereby to the bound on the size of G that is the conclusion of the lemma. The complete proof assembles a series of claims made against the witness tree, various sets of vertices, and inductive priorities and sets up a master inequality leading to a proof by induction, and a $3.5k$ problem kernel.

Objective B: Polynomial-Time Preprocessing and Data-Reduction Routines

The authors have designed a table for tracing each possible *boundary state* for a possible solution. Examples are given that show the surprising power of cascading data-reduction rules on real input distributions and that describe a variety of mathematical phenomena relating to reduction rules. For example, some reduction rules, such as the *Kleitman–West dissolver rule* for MAX LEAF (Fig. 2), have a fixed "boundary size" (in this case 2), whereas crown-type reduction rules do not have a fixed boundary size.

Objective C: Gradients and Solution Transformations for Local Search

A generalization of the usual setup for local search is given, based on the mathematical power of the more complicated gradient in obtaining superior kernelization bounds. Idea 1 is that local search be conducted based on maintaining a "current witness structure" rather than a full solution (spanning tree). Idea 2 is to use the list of inductive priorities to define a "better solution" gradient for the local search.

Objective D: Polynomial-Time Approximation Algorithms

The polynomial-time extremal structure theory leads directly to a constant-factor p-time approximation algorithm for MAX LEAF. First, reduce G using the kernelization rules. The rules are approximation-preserving. Take *any* tree T (not necessarily spanning) in G. If all of the structural claims hold, then (by the boundary lemma arguments) the tree T must have at least n/c leaves for $c = 3.75$. Therefore, lifting T back along the reduction path, we obtain a c-approximation.

If at least one of the structural claims does not hold, then the tree T can be improved against one of the inductive priorities. Notice that each claim is proved by an argument that can be interpreted as a polynomial-time routine that improves T, when the claim is contradicted.

These consequences can be applied to the original T (and its successors) only a polynomial number of times (determined by the list of inductive priorities) until one arrives at a tree T' for which all of the various structural claims hold. At that point, we must have a c-approximate solution.

Objective E: Structure To Exploit in The Ecology of Complexity

The objective here is to understand how every input-governing problem parameter affects the complexity of every other problem. As a small example, consider Table 1

Max Leaf Spanning Tree, Table 1
The complexity ecology of parameters

	TW	BW	VC	DS	G	ML
TW	*FPT*	W[1]-hard	*FPT*	*FPT*	?	*FPT*
BW	*FPT*	W[1]-hard	*FPT*	*FPT*	?	*FPT*
VC	*FPT*	?	*FPT*	*FPT*	?	*FPT*
DS	?	?	W[1]-hard	W[1]-hard	?	?
G	W[1]-hard	W[1]-hard	W[1]-hard	W[1]-hard	*FPT*	?
ML	*FPT*	?	*FPT*	*FPT*	*FPT*	?

using the shorthand TW is TREEWIDTH, BW is BANDWIDTH, VC is VERTEX COVER, DS is DOMINATING SET, G is GENUS and ML is MAX LEAF. The entry in the second row and fourth column indicates that there is an *FPT* algorithm to optimally solve the DOMINATING SET problem for a graph G of bandwidth at most k. The entry in the fourth row and second column indicates that it is unknown whether BANDWIDTH can be solved optimally by an *FPT* algorithm when the parameter is a bound on the domination number of the input.

MAX LEAF applies to the last row of the table. For graphs of *max leaf number* bounded by k, the maximum size of an independent set can be computed in time $O^*(2.972^k)$ based on a reduction to a kernel of size at most $7k$. There is a practical payoff for using the output of one problem as the input to another.

Applications

The MAX LEAF SPANNING TREE problem has motivations in computer graphics for creating triangle strip representations for fast interactive rendering [5]. Other applications are found in the area of traffic grooming and network design, such as the design of optical networks and the utilization of wavelengths in order to minimize network cost, either in terms of the line-terminating equipment deployed or in terms of electronic switching [6]. The minimum-energy problem in wireless networks consists of finding a transmission radius vector for all stations in such a way that the total transmission power of the whole network is the least possible. A restricted version of this problem is equivalent to the MAX LEAF SPANNING TREE problem [7]. Finding spanning trees with many leaves is equivalent to finding small connected dominating sets and is also called the MINIMUM CONNECTED DOMINATING problem [13].

Open Problems

Branching Strategies

While extremal structure is in some sense *the right way* to design an FPT algorithm, this is not the only way. In particular, the recipe is silent on what to do with the kernel. An open problem is to find general strategies for employing "parameter-appropriate structure theory" in branching strategies for sophisticated problem kernel analysis.

Turing Kernelizability

The polynomial-time transformation of (x, k) to the simpler reduced instance (x', k') is a many:1 transformation. One can generalize the notion of many:1 reduction to Turing reduction. How should the quest for p-time extremal theory unfold under this "more generous" FPT?

Algorithmic Forms of The Boundary Lemma Approach

The hypothesis of the boundary lemma that (G, k) is a *yes*-instance implies that there exists a witness structure to this fact. There is no assumption that one has algorithmic access to this structure, and when reduction rules are discovered, these have to be transformations that can be applied to (G, k) and a structure that can be discovered in (G, k) in polynomial time. In other words, reduction rules cannot be *defined* with respect to the witness structure. Is it possible to describe more general approaches to kernelization where the witness structure used in the proof of the boundary lemma is polynomial-time computable, and this structure provides a conditional context for some reduction rules? How would this change the extremal method recipe?

Problem Annotation

One might consider a generalized MAX LEAF problem where vertices and edges have various annotations as to whether they *must* be leaves (or internal vertices) in a solution, etc. Such a generalized form of the problem would generally be expected to be "more difficult" than the vanilla form of the problem. However, several of the "best known" FPT algorithms for various problems, are based on these generalized, annotated forms of the problems. Examples include PLANAR DOMINATING SET and FEEDBACK VERTEX SET [4]. Should annotation be part of the recipe for the best possible polynomial-time kernelization?

Cross References

► Connected Dominating Set
► Data Reduction for Domination in Graphs

Recommended Reading

1. Bonsma, P.: Spanning trees with many leaves: new extremal results and an improved FPT algorithm. Memorandum Department of Applied Mathematics, vol. 1793, University of Twente, Enschede (2006)
2. Bonsma, P., Brueggemann, T., Woeginger, G.: A faster FPT algorithm for finding spanning trees with many leaves. Proceedings of MFCS 2003. Lecture Notes in Computer Science, vol. 2747, pp. 259–268. Springer, Berlin (2003)
3. Downey, R.G., Fellows, M.R.: Parameterized complexity. Monographs in Computer Science. Springer, New York (1999)
4. Dehne, F., Fellows, M., Langston, M., Rosamond, F., Stevens, K.: An $O(2^{O(k)} n^3)$ FPT algorithm for the undirected feedback vertex set problem. Proceedings COCOON 2005. Lecture Notes in Computer Science, vol. 3595, pp. 859–869. Springer, Berlin (2005)
5. Diaz-Gutierrez, P., Bhushan, A., Gopi, M., Pajarola, R.: Single-strips for fast interactive rendering. J. Vis. Comput. **22**(6), 372–386 (2006)
6. Dutta, R., Savage, C.: A Note on the Complexity of Converter Placement Supporting Broadcast in WDM Optical Networks. In: Proceedings of the International Conference on Telecommunication Systems-Modeling and Analysis, Dallas, November 2005 ISBN: 0-9716253-3-6 pp. 23–31. American Telecommunication Systems Management Association, Nashville
7. Egecioglu, O., Gonzalez, T.: Minimum-energy Broadcast in Simple Graphs with Limited Node Power. In: Proc. IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS 2001), Anaheim, August 2001 pp. 334–338
8. Estivill-Castro, V., Fellows, M.R., Langston, M.A., Rosamond, F.A.: FPT is P-time extremal structure I. In: Algorithms and complexity in Durham 2005. Texts in Algorithmics, vol. 4, pp. 1–41. Kings College Publications, London (2005)
9. Fellows, M., Langston, M.: On well-partial-order theory and its applications to combinatorial problems of VLSI design. SIAM J. Discret. Math. **5**, 117–126 (1992)
10. Fellows, M.: Blow-ups, win/win's and crown rules: some new directions in FPT. In: Proceedings of the 29th Workshop on Graph Theoretic Concepts in Computer Science (WG 2003). Lecture Notes in Computer Science, vol. 2880, pp. 1–12. Springer, Berlin (2003)
11. Fellows, M., McCartin, C., Rosamond, F., Stege, U.: Coordinatized kernels and catalytic reductions: an improved FPT algorithm for max leaf spanning tree and other problems. In: Proceedings of the 20th Conference on Foundations of Software Technology and Theoretical Computer Science (FST-TCS 2000). Lecture Notes in Theoretical Computer Science 1974, pp. 240–251. Springer, Berlin (2000)
12. Kleitman, D.J., West, D.B.: Spanning trees with many leaves. SIAM J. Discret. Math. **4**, 99–106 (1991)
13. Kouider, M., Vestergaard, P.D.: Generalized connected domination in graphs. Discret. Math. Theor. Comput. Sci. (DMTCS) **8**, 57–64 (2006)
14. Lu, H.-I., Ravi, R.: Approximating maximum leaf spanning trees in almost linear time. J. Algorithm **29**, 132–141 (1998)
15. Niedermeier, R.: Invitation to Fixed Parameter Algorithms. Lecture Series in Mathematics and Its Applications, Oxford University Press, Oxford (2006)
16. Prieto-Rodriguez, E.: Systematic kernelization in FPT algorithm design. Dissertation, School of Electrical Engineering and Computer Science, University of Newcastle, Australia (2005)
17. Solis-Oba, R.: 2-approximation algorithm for finding a spanning tree with the maximum number of leaves. In: Proceedings of the 6th Annual European Symposium on Algorithms (ESA'98). Lecture Notes in Computer Science, vol. 1461, pp. 441–452. Springer, Berlin (1998)

Metrical Task Systems

1992; Borodin, Linial, Saks

MANOR MENDEL
Department of Mathematics and Computer Science, The Open University of Israel, Raanana, Israel

Keywords and Synonyms

MTS

Problem Definition

Metrical task systems (MTS), introduced by Borodin, Linial, and Saks [5], is a cost minimization problem defined on a metric space (X, d_X) and informally described as follows: A given *system* has a set of internal states X. The aim of the system is to serve a given sequence of tasks. The servicing of each task has a certain cost that depends on the task and the state of the system. The system may switch states before serving the task, and the total cost for servicing the task is the sum of the service cost of the task in the new state and the distance between the states in a metric space defined on the set of states. Following Manasse, McGeoch, and Sleator [11], an extended model is considered here, in which the set of allowable tasks may be restricted.

Notation

Let T^* denote the set of finite sequences of elements from a set T. For $x, y \in T^*$, $x \circ y$ is the concatenation of the sequences x and y, and $|x|$ is the length of the sequence x.

Definition 1 (Metrical Task System) Fix a metric space (X, d_X). Let $\Gamma = \{(r_x)_{x \in X} : \forall x \in X,\ r(x) \in [0, \infty]\}$ be the set of all possible tasks. Let $T \subseteq \Gamma$ be a subset of tasks, called *allowable tasks*.
MTS$((X, d_X), T, a_0 \in X)$:
INPUT: A finite sequence of tasks $\tau = (\tau_1, \dots, \tau_m) \in T^*$.
OUTPUT: A sequence of points $a = (a_1, \dots, a_m) \in X^*$, $|a| = |\tau|$.
OBJECTIVE: minimize

$$\mathsf{cost}(\tau, a) = \sum_{i=1}^{m} (d_X(a_{i-1}, a_i) + \tau_i(a_i)).$$

When $T = \Gamma$, the MTS problem is called *general*.

When X is finite and the task sequence $\tau \in T^*$ is given in advance, a dynamic programming algorithm can compute an optimal solution in space $O(|X|)$ and time $O(|\tau| \cdot |X|)$. MTS, however, is most interesting in an online setting, where the system must respond to a task τ_i with a state $a_i \in X$ without knowing the future tasks in τ. Formally,

Definition 2 (Online algorithms for MTS) A deterministic algorithm for a MTS$((X, d_X), T, a_0)$ is a mapping $S : T^* \to X^*$ such that for every $\tau \in T$, $|S(\tau)| = |\tau|$. A deterministic algorithm $S : T^* \to X^*$ is called *online* if for every $\tau, \sigma \in T^*$, there exists $a \in X^*$, $|a| = |\sigma|$ such that $S(\tau \circ \sigma) = S(\tau) \circ a$. A randomized online algorithm is a probability distribution over deterministic online algorithms.

Online algorithms for MTS are evaluated using *(asymptotic) competitive analysis*, which is, roughly speaking, the worst ratio of the algorithm's cost to the optimal cost taken over all possible task sequences.

Definition 3 A randomized online algorithm R for MTS$((X, d_X), a_0)$ is called c-competitive (against oblivious adversaries) if there exists $b = b(X) \in \mathbb{R}$ such that for any task sequence $\tau \in T^*$, and any point sequence $a \in X^*$, $|a| = |\tau|$,

$$\mathbb{E}[\mathsf{cost}(\tau, R(\tau))] \leq c \cdot \mathsf{cost}(\tau, a) + b,$$

where the expectation is taken over the distribution R.

The competitive ratio of an online algorithm R is the infimum over $c \geq 1$ for which R is c-competitive. The deterministic [respectively, randomized] competitive ratio of MTS$((X, d_X), T, a_0)$ is the infimum over the competitive ratios of all deterministic [respectively, randomized] online algorithms for this problem. Note that because of the existential quantifier over b, the asymptotic competitive ratio (both randomized and deterministic) of a MTS$((X, d_X), T, a_0)$ is independent of a_0, and it can therefore be dropped from the notation.

Key Results

Theorem 1 ([5]) *The deterministic competitive ratio of the general MTS problem on any n-point metric space is* $2n - 1$.

In contrast to the deterministic case, the understanding of randomized algorithms for general MTS is not complete, and generally no sharp bounds such as Theorem 1 are known.

Theorem 2 ([5,10]) *The randomized competitive ratio of the general MTS problem on n-point uniform space (where all distances are equal) is at least* $H_n = \sum_{i=1}^{n-1} i^{-1}$, *and at most* $(1 + o(1))H_n$.

The best bounds currently known for general n-point metrics are proved in two steps: First the given metric is approximated by an *ultrametric*, and then a bound on the competitive ratio of general MTS on ultrametrics is proved.

Theorem 3 ([8,9]) *For any n-point metric space* (X, d_X), *there exists an* $O(\log^2 n \log\log n)$ *competitive randomized algorithm for the general MTS on* (X, d_X).

The metric approximation component in the proof of Theorem 3 is called *probabilistic embedding*. An optimal $O(\log n)$ probabilistic embedding is shown by

Fakcheroenphol, Rao and Talwar before [8] improving on results by Alon, Karp, Peleg, and West and by Bartal, where this notion was invented. A different type of metric approximation with better bounds for metrics of low *aspect ratio* is given in [3].

Fiat and Mendel [9] show a $O(\log n \log\log n)$ competitive algorithm for n-point ultrametrics, improving (and using) a result of Bartal, Blum, Burch, and Tomkins [1], where the first poly-logarithmic (or even sublinear) competitive randomized algorithm for general MTS on general metric spaces is presented.

Theorem 4 ([2,12]) *For any n-point metric space (X, d_X), the randomized competitive ratio of the general MTS on (X, d_X) is at least $\Omega(\log n/\log\log n)$.*

The metric approximation component in the proof of Theorem 4 is called *Ramsey subsets*. It was first used in this context by Karloff, Rabani, and Ravid, later improved by Blum, Karloff, Rabani and Saks, and Bartal, Bollobás, and Mendel [2]. A tight result on Ramsey subsets is proved by Bartal, Linial, Mendel, and Naor. For a simpler (and stronger) proof, see [12].

A lower bound of $\Omega(\log n/\log\log n)$ on the competitive ratio of any randomized algorithm for general MTS on n-point ultrametrics is proved in [2], improving previous results of Karloff, Rabani, and Ravid, and Blum, Karloff, Rabani and Saks.

The last theorem is the only one not concerning general MTSs.

Theorem 5 ([6]) *It is PSPACE hard to determine the competitive ratio of a given MTS instance $((X, d_X), a_0 \in X, T)$, even when d_X is the uniform metric. On the other hand, when d_X is uniform, there is a polynomial time deterministic online algorithm for MTS$((X, d_X), a_0 \in X, T)$ whose competitive ratio is $O(\log|X|)$ times the deterministic competitive ratio of the MTS$((X, d_X), a_0, T)$. Here it is assumed that the instance $((X, d_X), a_0, T)$ is given explicitly.*

Applications

Metrical task systems were introduced as an abstraction for online computation, they generalize many concrete online problems such as paging, weighted caching, k-server, and list update. Historically, it served as an indicator for a general theory of competitive online computation.

The main technical contribution of the MTS model is the development of the work function algorithm used to prove the upper bound in Theorem 1. This algorithm was later analyzed by Koutsoupias and Papadimitriou in the context of the k-server problem, and was shown to be $2k-1$ competitive. Furthermore, although the MTS model generalizes the k-server problem, the general MTS problem on the n-point metric is essentially equivalent to the $(n-1)$-server problem on the same metric [2]. Hence, lower bounds on the competitive ratio of general MTS imply lower bounds for the k-server problem, and algorithms for general MTS may constitute a first step in devising an algorithm for the k-server problem, as is the case with the work function algorithm.

The metric approximations used in Theorem 3, and Theorem 4 have found other algorithmic applications.

Open Problems

There is still an obvious gap between the upper bound and lower bound known on the randomized competitive ratio of general MTS on general finite metrics. It is known that, contrary to the deterministic case, the randomized competitive ratio is *not* constant across all metric spaces of the same size. However, in those cases where exact bounds are known, the competitive ratio is $\Theta(\log n)$. An obvious conjecture is that the randomized competitive is $\Theta(\log n)$ for any n-point metric. Arguably, the simplest classes of metric spaces for which no upper bound on the randomized competitive ratio better than $O(\log^2 n)$ is known, are paths and cycles.

Also lacking is a "middle theory" for MTS. On the one hand, general MTS are understood fairly well. On the other hand, specialized MTS such as list update, deterministic k-server algorithms, and deterministic weighted-caching, are also understood fairly well, and have a much better competitive ratio than the corresponding general MTS. What may be missing are "in between" models of MTS that can explain the low competitive ratios for some of the concrete online problems mentioned above.

It would be also nice to strengthen Theorem 5, and obtain a polynomial time deterministic online algorithm whose competitive ratio on any MTS instance on *any* n-point metric space is at most poly-log(n) times the deterministic competitive ratio of that MTS instance.

Cross References

- ▶ Algorithm DC-Tree for k Servers on Trees
- ▶ Approximating Metric Spaces by Tree Metrics
- ▶ Online List Update
- ▶ Online Paging and Caching
- ▶ Paging
- ▶ Ski Rental Problem
- ▶ Work-Function Algorithm for k Servers

Recommended Reading

1. Bartal, Y., Blum, A., Burch, C., Tomkins, A.: A polylog()-competitive algorithm for metrical task systems. In: Proceedings of the 29th annual ACM Symposium on the Theory of Computing, pp. 711–719. ACM, New York (1997)
2. Bartal, Y., Bollobás, B., Mendel, M.: Ramsey-type theorems for metric spaces with applications to online problems. J. Comput. Syst. Sci. **72**, 890–921 (2006)
3. Bartal, Y., Mendel, M.: Multiembedding of metric spaces. SIAM J. Comput. **34**, 248–259 (2004)
4. Borodin, A., El-Yaniv, R.: Online computation and competitive analysis. Cambridge University Press, Cambridge, UK (1998)
5. Borodin, A., Linial, N., Saks, M.E.: An optimal on-line algorithm for metrical task system. J. ACM **39**, 745–763 (1992)
6. Burley, W.R., Irani, S.: On algorithm design for metrical task systems. Algorithmica **18**, 461–485 (1997)
7. Chrobak, M., Larmore, L.L.: Metrical task systems, the server problem and the work function algorithm. In: Fiat, A., Woeginger, G.J. (eds.) Online Algorithms. The State of the Art. LNCS, vol. 1442, ch. 4, pp. 74–96. Springer, London (1998)
8. Fakcharoenphol, J., Rao, S., Talwar, K.: A tight bound on approximating arbitrary metrics by tree metrics. J. Comput. Syst. Sci. **69**, 485–497 (2004)
9. Fiat, A., Mendel, M.: Better algorithms for unfair metrical task systems and applications. SIAM J. Comput. **32**, 1403–1422 (2003)
10. Irani, S., Seiden, S.S.: Randomized algorithms for metrical task systems. Theor. Comput. Sci. **194**, 163–182 (1998)
11. Manasse, M.S., McGeoch, L.A., Sleator, D.D.: Competitive algorithms for server problems. J. Algorithms **11**, 208–230 (1990)
12. Mendel, M., Naor, A.: Ramsey partitions and proximity data structures. J. Eur. Math. Soc. **9**(2), 253–275 (2007)

Metric TSP

1976; Christofides

Markus Bläser
Department of Computer Science, Saarland University, Saarbrücken, Germany

Keywords and Synonyms

Metric traveling salesman problem; Metric traveling salesperson problem

Problem Definition

The *Traveling Salesman Problem (TSP)* is the following optimization problem:

Input: A complete loopless undirected graph $G = (V, E, w)$ with a weight function $w\colon E \to \mathbb{Q}_{\geq 0}$ that assigns to each edge a non-negative weight.

Feasible solutions: All Hamiltonian tours, i. e, the subgraphs H of G that are connected, and each node in them that has degree two.

Objective function: The weight function $w(H) = \sum_{e \in H} w(e)$ of the tour.

Goal: Minimization.

The TSP is an NP-hard optimization problem. This means that a polynomial time algorithm for the TSP does not exist unless P = NP. One way out of this dilemma is provided by *approximation algorithms*. A polynomial time algorithm for the TSP is called an α-approximation algorithm if the tour H produced by the algorithm fulfills $w(H) \leq \alpha \cdot \mathrm{OPT}(G)$. Here $\mathrm{OPT}(G)$ is the weight of a minimum weight tour of G. If G is clear from the context, one just writes OPT. An α-approximation algorithm always produces a feasible solution whose objective value is at most a factor of α away from the optimum value. α is also called the approximation factor or performance guarantee. α does not need to be a constant; it can be a function that depends on the size of the instance or the number of nodes n.

If there exists a polynomial time approximation algorithm for the TSP that achieves an exponential approximation factor in n, then P = NP [6]. Therefore, one has to look at restricted instances. The most natural restriction is the *triangle inequality*, that means,

$$w(u, v) \leq w(u, x) + w(x, v) \quad \text{for all } u, v, x \in V\,.$$

The corresponding problem is called the *Metric TSP*. For the Metric TSP, approximation algorithms that achieve a constant approximation factor exist. Note that for the Metric TSP, it is sufficient to find a tour that visits each vertex *at least* once: Given such a tour, we can find a Hamiltonian tour of no larger weight by skipping every vertex that we already visited. By the triangle inequality, the new tour cannot get heavier.

Key Results

A simple 2-approximation algorithm for the Metric TSP is the *tree doubling algorithm*. It uses minimum spanning trees to compute Hamiltonian tours. A *spanning tree* T of a graph $G = (V, E, w)$ is a connected acyclic subgraph of G that contains each node of V. The weight $w(T)$ of such a spanning tree is the sum of the weights of the edges in it, i. e., $w(T) = \sum_{e \in T} w(e)$. A spanning tree is called a minimum spanning tree if its weight is minimum among all spanning trees of G. One can efficiently compute a minimum spanning tree, for instance via Prim's or Kruskal's algorithm, see e. g. [5].

The tree doubling algorithm seems to be folklore. The next lemma is the key for proving the upper bound on the approximation performance of the tree doubling algorithm.

```
Input: a complete loopless edge weighted undirected
    graph $G = (V, E, w)$ with weight function $w : E \to \mathbb{Q}_{\geq 0}$ that fulfills the triangle inequality
Output: a Hamiltonian tour of $G$ that is a 2" approximation
1: Compute a minimum spanning tree $T$ of $G$.
2: Duplicate each edge of $T$ and obtain a Eulerian multigraph $T'$.
3: Compute a Eulerian tour of $T'$ (for instance via a depth first search in $T$). Whenever a node is visited in the Eulerian tour that was already visited, this node is skipped and one proceeds with the next unvisited node along the Eulerian cycle. (This process is called shortcutting.) Return the resulting Hamiltonian tour $H$.
```

Metric TSP, Algorithm 1
Tree doubling algorithm

Lemma 1 *Let T be a minimum spanning tree of $G = (V, E, w)$. Then $w(T) \leq \mathrm{OPT}$.*

Proof If one deletes any edge of a Hamiltonian tour of G, one gets a spanning tree of G. □

Theorem 2 *Algorithm 1 always returns a Hamiltonian tour whose weight is at most twice the weight of an optimum tour. Its running time is polynomial.*

Proof By Lemma 1, $w(T) \leq \mathrm{OPT}$. Since one duplicates each edge of T, the weight of T' equals $w(T') = 2w(T) \leq 2\mathrm{OPT}$. When taking shortcuts in step 3, a path in T' is replaced by a single edge. By the triangle inequality, the sum of the weights of the edges in such a path is at least the weight of the edge it is replaced by. (Here, the algorithm breaks down for arbitrary weight functions.) Thus $w(H) \leq w(T')$. This proves the claim about the approximation performance.

The running time is dominated by the time needed to compute a minimum spanning tree. This is clearly polynomial. □

Christofides' algorithm (Algorithm 2) is a clever refinement of the tree doubling algorithm. It first computes a minimum spanning tree. On the nodes that have an odd degree in T, it then computes a minimum weight perfect matching. A matching M of G is called a matching on $U \subseteq V$ if all edges of M consist of two nodes from U. Such a matching is called *perfect* if every node of U is incident with an edge of M.

Lemma 3 *Let $U \subseteq V, \#U$ even. Let M be a minimum weight perfect matching on U. Then $w(M) \leq \mathrm{OPT}/2$.*

```
Input: a complete loopless edge weighted undirected
    graph $G = (V, E, w)$ with weight function $w : E \to \mathbb{Q}_{\geq 0}$ that fulfills the triangle inequality
Output: a Hamiltonian tour of $G$ that is a 3/2" approximation
1: Compute a minimum spanning tree $T$ of $G$.
2: Let $U \subseteq V$ be the set of all nodes that have odd degree in $T$. In $G$, compute a minimum weight perfect matching $M$ on $U$.
3: Compute a Eulerian tour of $T \cup M$ (considered as a multigraph).
4: Take shortcuts in this Eulerian tour to a Hamiltonian tour $H$.
```

Metric TSP, Algorithm 2
Christofides' algorithm

Proof Let H be an optimum Hamiltonian tour of G. One takes shortcuts in H to get a tour H' on $G|_U$ as follows: H induces a permutation of the nodes in U, namely the order in which the nodes are visited by H. One connects the nodes of U in the order given by the permutation. To every edge of H' corresponds a path in H connecting the two nodes of this edge. By the triangle inequality, $w(H') \leq w(H)$. Since $\#U$ is even, H' is the union of two matchings. The lighter one of these two has a weight of at most $w(H')/2 \leq \mathrm{OPT}/2$. □

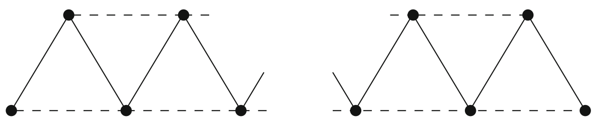
One can compute a minimum weight perfect matching in time $O(n^3)$, see for instance [5].

Theorem 4 *Algorithm 2 is a 3/2-approximation algorithm with polynomial running time.*

Proof First observe that the number of odd degree nodes of the spanning tree is even, since the sum of the degrees of all nodes equals $2(n-1)$, which is even. Thus a perfect matching on U exists. The weight of the Eulerian tour is obviously $w(T) + w(M)$. By Lemma 1, $w(T) \leq \mathrm{OPT}$. By Lemma 3, $w(M) \leq \mathrm{OPT}/2$. The weight $w(H)$ of the computed tour H is at most the weight of the Eulerian tour by the triangle inequality, i. e., $w(H) \leq \frac{3}{2}\mathrm{OPT}$. Thus the algorithm is a 3/2-approximation algorithm. Its running time is $O(n^3)$. □

Applications

Experimental analysis shows that Christofides' algorithm itself deviates by 10% to 15% from the optimum tour [3]. However, it can serve as a good starting tour for other heuristics like the Lin–Kernigham heuristic.



Metric TSP, Figure 1

A tight example for Christofides' algorithm. There are $2n+1$ nodes. *Solid edges* have a weight of one, *dashed ones* have a weight of $1+\epsilon$

Open Problems

The analysis of Algorithm 2 is tight; an example is the metric completion of the graph depicted in Fig. 1. The unique minimum spanning tree consists of all solid edges. It has only two nodes of odd degree. The edge between these two nodes has weight $(1+\epsilon)(n+1)$. No shortcuts are needed, and the weight of the tour produced by the algorithm is $\approx 3n$. An optimum tour consists of all dashed edges plus the leftmost and rightmost solid edge. The weight of this tour is $(2n-1)(1+\epsilon)+2 \approx 2n$.

The question whether there is an approximation algorithm with a better performance guarantee is a major open problem in the theory of approximation algorithms.

Held and Karp [2] design an LP based algorithm that computes a lower bound for the weight of an optimum TSP tour. It is conjectured that the weight of an optimum TSP tour is at most a factor of 4/3 larger than this lower bound, but this conjecture is unproven for more than three decades. An algorithmic proof of this conjecture would yield an 4/3-approximation algorithm for the Metric TSP.

Experimental Results

See e. g. [3], where a deviation of 10% to 15% of the optimum (more precisely of the Held–Karp bound) is reported for various sorts of instances.

Data Sets

The webpage of the 8th DIMACS implementation challenge, www.research.att.com/~dsj/chtsp/, contains a lot of instances.

Cross References

▶ Minimum Spanning Trees

Recommended Reading

Christofides never published his algorithm. It is usually cited as one of two technical reports from Carnegie Mellon University, TR 388 of the Graduate School of Industrial Administration (now Tepper School of Business) and CS-93-13. None of them seem to be available at Carnegie Mellon University anymore [Frank Balbach, personal communication, 2006]. A one-page abstract was published in a conference record. But his algorithm quickly found his way into standard textbooks on algorithm theory, see [7] for a recent one.

1. Christofides, N.: Worst case analysis of a new heuristic for the traveling salesman problem, Technical Report 388, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, (1976). Also: Carnegie-Mellon University Technical Report CS-93-13, 1976. Abstract in Traub, J.F. (ed.) Symposium on new directions and recent results in algorithms and complexity, pp. 441. Academic Press, New York (1976)
2. Held, M., Karp, R.M.: The traveling salesman problem and minimum spanning trees. Oper. Res. **18**, 1138–1162 (1970)
3. Johnson, D.S., McGeoch, L.A.: Experimental analysis of heuristics for the STSP. In: Gutin, G., Punnen, A.P. (eds.) The Traveling Salesman Problem and its Variations. Kluwer, Dordrecht (2002)
4. Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., Shmoys, D.B. (eds.): The Traveling Salesman Problem. A Guided Tour of Combinatorial Optimization. Wiley, Chichester (1985)
5. Papadimitriou, C., Steiglitz, K.: Combinatorial Optimization: Algorithms and Complexity. Prentice-Hall, Englewood Cliffs (1982)
6. Sahni, S., Gonzalez, T.: P-complete approximation problems. J. ACM **23**, 555–565 (1976)
7. Vazirani, V.V.: Approximation Algorithms. Springer, Berlin (2001)
8. Traveling Salesman Problem. www.tsp.gatech.edu (2006). Accessed 28 Mar 2008

Minimum Bisection

1999; Feige, Krauthgamer

ROBERT KRAUTHGAMER[1,2]
[1] Weizmann Institute of Science, Rehovot, Israel
[2] IBM Almaden Research Center, San Jose, CA, USA

Keywords and Synonyms

Graph bisection

Problem Definition

Overview

Minimum bisection is a basic representative of a family of discrete optimization problems dealing with partitioning the vertices of an input graph. Typically, one wishes to minimize the number of edges going across between the different pieces, while keeping some control on the partition, say by restricting the number of pieces and/or their size. (This description corresponds to an edge-cut of the graph; other variants correspond to a vertex-cut with similar restrictions.) In the minimum bisection problem, the goal is to partition the vertices of an input graph into two equal-size sets, such that the number of edges connecting the two sets is as small as possible.

In a seminal paper in 1988, Leighton and Rao [14] devised for MINIMUM-BISECTION a logarithmic-factor bicriteria approximation algorithm.[1] Their algorithm has found numerous applications, but the question of finding a true approximation with a similar factor remained open for over a decade later. In 1999, Feige and Krauthgamer [6] devised the first polynomial-time algorithm that approximates this problem within a factor that is polylogarithmic (in the graph size).

Cuts and Bisections

Let $G = (V, E)$ be an undirected graph with $n = |V|$ vertices, and assume for simplicity that n is even. For a subset S of the vertices, let $\bar{S} = V \setminus S$. The *cut* (also known as *cutset*) $(S, \bar{S})$ is defined as the set of all edges with one endpoint in S and one endpoint in $\bar{S}$. These edges are said to *cross* the cut, and the two sets S and $\bar{S}$ are called the two *sides* of the cut.

Assume henceforth that G has nonnegative edge-weights. (In the unweighted version, every edge has a unit weight.) The *cost* of a cut $(S, \bar{S})$ is then defined to be the total edge-weight of all the edges crossing the cut.

A cut $(S, \bar{S})$ is called a *bisection* of G if its two sides have equal cardinality, namely $|S| = |\bar{S}| = n/2$. Let $b(G)$ denote the minimum cost of a bisection of G.

Problem 1 (MINIMUM-BISECTION)
Input: *An undirected graph G with nonnegative edge-weights.*
Output: *A bisection $(S, \bar{S})$ of G that has minimum cost.*

This definition has a crucial difference from the classical MINIMUM-CUT problem (see e. g. [10] and references therein), namely, there is a restriction on the sizes of the two sides of the cut. As it turns out, MINIMUM-BISECTION is NP-hard (see [9]), while MINIMUM-CUT can be solved in polynomial time.

Balanced Cuts and Edge Separators

The above rather basic definition of minimum bisection can be extended in several ways. Specifically, one may require only an upper bound on the size of each side. For $0 < \beta < 1$, a cut $(S, \bar{S})$ is called β-balanced if $\max\{|S|, |\bar{S}|\} \leq \beta n$. Note the latter requirement implies $\min\{|S|, |\bar{S}|\} \geq (1 - \beta)n$. In this terminology, a bisection is a 1/2-balanced cut.

Problem 2 (β-BALANCED-CUT)
Input: *An undirected graph G with nonnegative edge-weights.*
Output: *A β-balanced cut $(S, \bar{S})$ of G with $\max\{|S|, |\bar{S}|\} \leq \beta n$, that has cost as small as possible.*

The special case of $\beta = 2/3$ is commonly refered to as the EDGE-SEPARATOR problem.

In general, the sizes of the two sides may be specified in advance arbitrarily (rather than being equal); in this case the input contains a number k, and the goal is to find a cut $(S, \bar{S})$ such that $|S| = k$. One may also wish to divide the graph into more than two pieces of equal size and then the input contains a number $r \geq 2$, or alternatively, to divide the graph into r pieces of whose sizes are $k_1, \ldots, k_r$, where the numbers k_i are prescribed in the input; in either case, the goal is to minimize the number of edges crossing between different pieces.

Problem 3 (PRESCRIBED-PARTITION)
Input: *An undirected graph $G = (V, E)$ with nonnegative edge-weights, and integers $k_1, \ldots, k_r$ such that $\sum_i k_i = |V|$.*
Output: *A partition $V = V_1 \cup \cdots \cup V_r$ of G with $|V_i| = k_i$ for all i, such that the total edge-weight of edges whose endpoints lie in different sets V_i is as small as possible.*

Key Results

The main result of Feige and Krauthgamer [6] is an approximation algorithm for MINIMUM-BISECTION. The approximation factor they originally claimed is $O(\log^2 n)$, because it used the algorithm of Leighton and Rao [14]; however, by using instead the algorithm of [2], the factor immediately improves to $O(\log^{1.5} n)$.

Theorem 1 Minimum-Bisection *can be approximated in polynomial time within $O(\log^{1.5} n)$ factor. Specifically, the algorithm produces for an input graph G a bisection $(S, \bar{S})$ whose cost is at most $O(\log^{1.5} n) \cdot b(G)$.*

The algorithm immediately extends to similar results for related and/or more general problems that are defined above.

Theorem 2 β-Balanced-Cut *(and in particular* Edge-Separator*) can be approximated in polynomial time within $O(\log^{1.5} n)$ factor.*

Theorem 3 Prescribed-Partition *can be approximated in time $n^{O(r)}$ to within $O(\log^{1.5} n)$ factor.*

For all three problems above, the approximation ratio improves to $O(\log n)$ for the family of graphs excluding

[1] A bicriteria approximation algorithm partitions the vertices into two sets each containing at most 2/3 of the vertices, and its value, i. e. the number of edges connecting the two sets, is compared against that of the best partition into equal-size sets.

a fixed minor (which includes in particular planar graphs). For simplicity, this result is stated for Minimum-Bisection.

Theorem 4 *In graphs excluding a fixed graph as a minor (e. g., planar graphs), the problems (i)* Minimum-Bisection*, (ii) β*-Balanced-Cut*, and (iii)* Prescribed-Partition *with fixed r can all be approximated in polynomial time within factor $O(\log n)$.*

It should be noted that all these results can be generalized further, including vertex-weights and terminals-vertices ($s - t$ pairs), see [Sect. 5 in 6].

Related Work

A bicriteria approximation algorithm for β-balanced cut returns a cut that is β'-balanced for a predetermined $\beta' > \beta$. For bisection, for example, $\beta = 1/2$ and typically $\beta' = 2/3$.

The algorithms in the above theorems use (in a black-box manner) an approximation algorithm for a problem called minimum quotient-cuts (or equivalently, sparsest-cut with uniform-demands). For this problem, the best approximation currently known is $O(\sqrt{\log n})$ for general graphs due to Arora, Rao, and Vazirani [2], and $O(1)$ for graphs excluding a fixed minor due to Klein, Plotkin, and Rao [13]. These approximation algorithms for minimum quotient-cuts immediately give a polynomial time bicriteria approximation (sometimes called pseudo-approximation) for Minimum-Bisection. For example, in general graphs the algorithm is guaranteed to produce a 2/3-balanced cut whose cost is at most $O(\sqrt{\log n}) \cdot b(G)$. Note however that a 2/3-balanced cut does not provide a good approximation for the value of $b(G)$. For instance, if G consists of three disjoint cliques of equal size, an optimal 2/3-balanced cut has no edges, whereas $b(G) = \Omega(n^2)$. For additional related work, including approximation algorithms for dense graphs, for directed graphs, and for other graph partitioning problems, see [Sect. 1 in 6] and the references therein.

Applications

One major motivation for Minimum-Bisection, and graph partitioning in general, is a divide-and-conquer approach to solving a variety of optimization problems, especially in graphs, see e. g. [15,16]. In fact, these problems arise naturally in a wide range of practical settings such as VLSI design and image processing; sometimes, the motivation is described differently, e. g. as a clustering task.

Another application of Minimum-Bisection is in assignment problems, of a form that is common in parallel systems and in scientific computing: jobs need to be assigned to machines in a balanced way, while assigning certain pairs of jobs the same machine, as much as possible. For example, consider assigning n jobs to 2 machines, when the amount of communication between every two jobs is known, and the goal is to have equal load (number of jobs) on each machine, and bring to minimum the total communication that goes between the machines. Clearly, this last problem can be restated as Minimum-Bisection in a suitable graph.

It should be noted that in many of these settings, a true approximation is not absolutely necessary, and a bicriteria approximation may suffice. Nevertheless, the algorithms stated in Sect. "Key Results" have been used to design algorithms for other problems, such as (1) an approximation algorithm for minimum bisection in k-uniform hypergraphs [3]; (2) an approximation algorithm for a variant of the minimum multicut problem [17]; and (3) an algorithm that efficiently certifies the unsatisfiability of random $2k$-SAT with sufficiently many clauses [5].

From a practical perspective, numerous heuristics (algorithms without worst-case guarantees) for graph partitioning have been proposed and studied, see [1] for an extensive survey. For example, one of the most famous heuristics is Kerninghan and Lin's local search heuristic for minimum bisection [11].

Open Problems

Currently, there is a large gap between the $O(\log^{1.5} n)$ approximation ratio for Minimum-Bisection achieved by Theorem 1 and the hardness of approximation results known for it. As mentioned above, Minimum-Bisection is known to be NP-hard (see [9]).

The problem is not known to be APX-hard but several results provide evidence towards this possibility. Bui and Jones [4] show that for every fixed $\epsilon > 0$, it is NP-hard to approximate the minimum bisection within an *additive* term of $n^{2-\epsilon}$. Feige [7] showed that if refuting 3SAT is hard on average on a natural distribution of inputs, then for every fixed $\varepsilon > 0$ there is no $4/3 - \varepsilon$ approximation algorithm for minimum bisection. Khot [12] proved that minimum bisection does not admit a polynomial-time approximation scheme (PTAS) unless NP has randomized sub-exponential time algorithms.

Taking a broader perspective, currently there is a (multiplicative) gap of $O(\log n)$ between the approximation ratio for Minimum-Bisection and that of minimum quotient-cuts (and thus also to the factor achieved by bicriteria approximation). It is interesting whether this gap can be reduced, e. g. by using the algorithm of [2] in a non-black box manner.

The vertex-cut version of MINIMUM-BISECTION is defined as follows: the goal is to partition the vertices of the input graph into $V = A \cup B \cup S$ with $|S|$ as small as possible, under the constraints that $\max\{|A|, |B|\} \leq n/2$ and no edge connects A with B. It is not known whether a polylogarithmic factor approximation can be attained for this problem. It should be noted that the same question regarding the directed version of MINIMUM-BISECTION was answered negatively by Feige and Yahalom [8].

Cross References

See entry on the paper by Arora, Rao, and Vazirani [2].

► Separators in Graphs

► Sparsest Cut

Recommended Reading

1. Alpert, C.J., Kahng, A.B.: Recent directions in netlist partitioning: a survey. Integr. VLSI J. **19**(1–2), 1–81 (1995)
2. Arora, S., Rao, S., Vazirani, U.: Expander flows, geometric embeddings, and graph partitionings. In: 36th Annual Symposium on the Theory of Computing, pp. 222–231, Chicago, June 2004
3. Berman, P., Karpinski, M.: Approximability of hypergraph minimum bisection. ECCC Report TR03-056, Electronic Colloquium on Computational Complexity, vol. 10 (2003)
4. Bui, T.N., Jones, C.: Finding good approximate vertex and edge partitions is NP-hard. Inform. Process. Lett. **42**(3), 153–159 (1992)
5. Coja-Oghlan, A., Goerdt, A., Lanka, A., Schädlich, F.: Techniques from combinatorial approximation algorithms yield efficient algorithms for random 2k-SAT. Theor. Comput. Sci. **329**(1–3), 1–45 (2004)
6. Feige, U., Krauthgamer, R.: A polylogarithmic approximation of the minimum bisection. SIAM Review **48**(1), 99–130 (2006) (Previous versions appeared in Proceedings of 41st FOCS, 1999; and in SIAM J. Comput. 2002)
8. Feige, U., Yahalom, O.: On the complexity of finding balanced oneway cuts. Inf. Process. Lett. **87**(1), 1–5 (2003)
7. Feige, U.: Relations between average case complexity and approximation complexity. In: 34th Annual ACM Symposium on the Theory of Computing, pp. 534–543, Montréal, May 19–21, 2002
9. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-completeness. W.H. Freeman and Company (1979)
10. Karger, D.R.: Minimum cuts in near-linear time. J. ACM **47**(1), 46–76 (2000)
11. Kernighan, B.W., Lin, S.: An efficient heuristic procedure for partitioning graphs. Bell Syst. Tech. J. **49**(2), 291–307 (1970)
12. Khot, S.: Ruling out PTAS for graph Min-Bisection, Densest Subgraph and Bipartite Clique. In: 45th Annual IEEE Symposium on Foundations of Computer Science, pp. 136–145, Georgia Inst. of Technol., Atlanta 17–19 Oct. 2004
13. Klein, P., Plotkin, S.A., Rao, S.: Excluded minors, network decomposition, and multicommodity flow. In: 25th Annual ACM Symposium on Theory of Computing, pp. 682–690, San Diego, 1993 May 16–18
14. Leighton, T., Rao, S.: Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. J. ACM **46**(6), 787–832, 29th FOCS, 1988 (1999)
15. Lipton, R.J., Tarjan, R.E.: Applications of a planar separator theorem. SIAM J. Comput. **9**(3), 615–627 (1980)
16. Rosenberg, A.L., Heath, L.S.: Graph separators, with applications. Frontiers of Computer Science. Kluwer Academic/Plenum Publishers, New York (2001)
17. Svitkina, Z., Tardos, É.: Min-Max multiway cut. In: 7th International workshop on Approximation algorithms for combinatorial optimization (APPROX), pp. 207–218, Cambridge, 2004 August 22–24

Minimum Congestion Redundant Assignments

2002; Fotakis, Spirakis

DIMITRIS FOTAKIS[1], PAUL SPIRAKIS[2]

[1] Department of Information and Communication Systems Engineering, University of the Aegean, Samos, Greece

[2] Computer Engineering and Informatics, Research and Academic Computer Technology Institute, Patras University, Patras, Greece

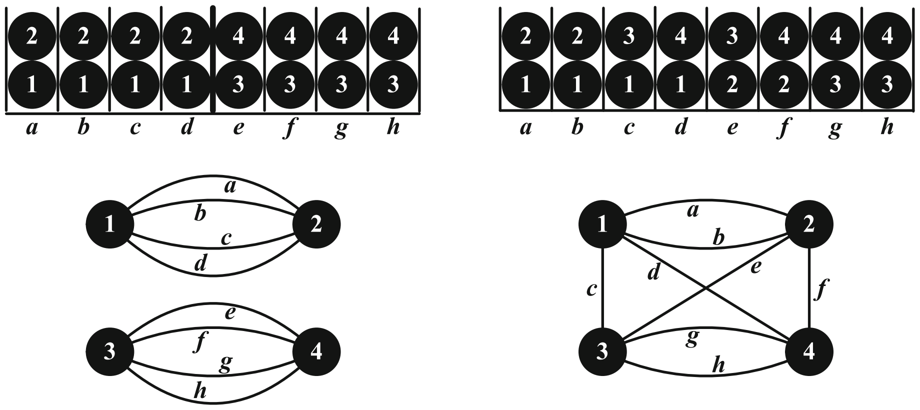
Keywords and Synonyms

Minimum fault-tolerant congestion; Maximum fault-tolerant partition

Problem Definition

This problem is concerned with the most efficient use of redundancy in load balancing on faulty parallel links. More specifically, this problem considers a setting where some messages need to be transmitted from a source to a destination through some faulty parallel links. Each link fails independently with a given probability, and in case of failure, none of the messages assigned to it reaches the destination[1]. An assignment of the messages to the links may use redundancy, i. e. assign multiple copies of some messages to different links. The reliability of a redundant assignment is the probability that every message has a copy

[1]This assumption is realistic if the messages are split into many small packets transmitted in a round-robin fashion. Then the successful delivery of a message requires that all its packets should reach the destination.



Minimum Congestion Redundant Assignments, Figure 1

Two redundant assignments of 4 unit size messages to 8 identical links. Both assign every message to 4 links and 2 messages to every link. The corresponding graph is depicted below each assignment. The assignment on the *left* is the most reliable 2-partitioning assignment ϕ_2. Lemma 3 implies that for every failure probability f, ϕ_2 is at least as reliable as any other assignment ϕ with $\mathrm{Cong}(\phi) \leq 2$. For instance, ϕ_2 is at least as reliable as the assignment on the *right*. Indeed the reliability of the assignment on the right is $1 - 4f^4 + 2f^6 + 4f^7 - 3f^8$, which is bounded from above by $\mathrm{Rel}(\phi_2) = 1 - 2f^4 + f^8$ for all $f \in [0, 1]$

on some active link, thus managing to reach the destination. Redundancy increases reliability, but also increases the message load assigned to the links. A good assignment should achieve high reliability and keep the maximum load of the links as small as possible.

The reliability of a redundant assignment depends on its structure. In particular, the reliability of different assignments putting the same load on every link and using the same number of copies for each message may vary substantially (e. g. compare the reliability of the assignments in Fig. 1). The crux of the problem is to find an efficient way of exploiting redundancy in order to achieve high reliability and low maximum load[2].

The work of Fotakis and Spirakis [1] formulates the scenario above as an optimization problem called *Minimum Fault-Tolerant Congestion* and suggests a simple and provably efficient approach of exploiting redundancy. This approach naturally leads to the formulation of another interesting optimization problem, namely that of computing an efficient fault-tolerant partition of a set of faulty parallel links. [1] presents polynomial-time approximation algorithms for computing a fault-tolerant partition of the links and proves that combining fault-tolerant partitions with standard load balancing algorithms results in a good approximation to Minimum Fault-Tolerant Congestion. To the best knowledge of the entry authors, this work is the first to consider the approximability of computing a redundant assignment that minimizes the maximum load of the links subject to the constraint that random faults should be tolerated with a given probability.

[2]If one does not insist on minimizing the maximum load, a reliable assignment is constructed by assigning every message to the most reliable links.

Notations and Definitions

Let M denote a set of m faulty parallel links connecting a source s to a destination t, and let J denote a set of n messages to be transmitted from s to t. Each link i has a rational capacity $c_i \geq 1$ and a rational failure probability $f_i \in (0, 1)$. Each message j has a rational size $s_j \geq 1$. Let $f_{\max} \equiv \max_{i \in M}\{f_i\}$ denote the failure probability of the most unreliable link. Particular attention is paid to the special case of identical capacity links, where all capacities are assumed to be equal to 1.

The reliability of a set of links M', denoted $\mathrm{Rel}(M')$, is the probability that there is an active link in M'. Formally, $\mathrm{Rel}(M') \equiv 1 - \prod_{i \in M'} f_i$. The reliability of a collection of disjoint link subsets $\mathcal{M} = \{M_1, \ldots, M_\nu\}$, denoted $\mathrm{Rel}(\mathcal{M})$, is the probability that there is an active link in every subset of $\mathcal{M}$. Formally,

$$\mathrm{Rel}(\mathcal{M}) \equiv \prod_{\ell=1}^{\nu} \mathrm{Rel}(M_\ell) = \prod_{\ell=1}^{\nu} \left(1 - \prod_{i \in M_\ell} f_i \right) .$$

A redundant assignment $\phi : J \mapsto 2^M \setminus \emptyset$ is a function that assigns every message j to a non-empty set of links $\phi(j) \subseteq M$. An assignment ϕ is feasible for a set of links M' if for every message j, $\phi(j) \cap M' \neq \emptyset$. The *reliability* of an assignment ϕ, denoted $\mathrm{Rel}(\phi)$, is the probability that ϕ is feasible for the actual set of active links. Formally,

$$\mathrm{Rel}(\phi) \equiv \sum_{\substack{M' \subseteq M \\ \forall j \in J,\, \phi(j) \cap M' \neq \emptyset}} \left(\prod_{i \in M'} (1 - f_i) \prod_{i \in M \setminus M'} f_i \right)$$

The *congestion* of an assignment ϕ, denoted $\mathrm{Cong}(\phi)$, is the maximum load assigned by ϕ to a link in M. Formally,

$$\mathrm{Cong}(\phi) \equiv \max_{i \in M} \left\{ \sum_{j:\, i \in \phi(j)} \frac{s_j}{c_i} \right\} .$$

Problem 1 (Minimum Fault-Tolerant Congestion)
INPUT: *A set of faulty parallel links $M = \{(c_1, f_1), \ldots, (c_m, f_m)\}$, a set of messages $J = \{s_1, \ldots, s_n\}$, and a rational number $\epsilon \in (0, 1)$.*
OUTPUT: *A redundant assignment $\phi : J \mapsto 2^M \setminus \emptyset$ with $Rel(\phi) \geq 1 - \epsilon$ that minimizes $Cong(\phi)$.*

Minimum Fault-Tolerant Congestion is **NP**-hard because it is a generalization of minimizing makespan on (reliable) parallel machines. The decision version of Minimum Fault-Tolerant Congestion belongs to **PSPACE**, but it is not clear whether it belongs to **NP**. The reason is that computing the reliability of a redundant assignment and deciding whether it is a feasible solution is **#P**-complete.

The work of Fotakis and Spirakis [1] presents polynomial-time approximation algorithms for Minimum Fault-Tolerant Congestion based on a simple and natural class of redundant assignments whose reliability can be computed easily. The high level idea is to separate the reliability aspect from load balancing. Technically, the set of links is partitioned in a collection of disjoint subsets $\mathcal{M} = \{M_1, \ldots, M_\nu\}$ with $\mathrm{Rel}(\mathcal{M}) \geq 1 - \epsilon$. Every subset $M_\ell \in \mathcal{M}$ is regarded as a *reliable link* of *effective capacity* $c(M_\ell) \equiv \min_{i \in M_\ell}\{c_i\}$. Then any algorithm for load balancing on reliable parallel machines can be used for assigning the messages to the subsets of $\mathcal{M}$, thus computing a redundant assignment ϕ with $\mathrm{Rel}(\phi) \geq 1 - \epsilon$.

The assignments produced by this approach are called *partitioning assignments*. More precisely, an assignment $\phi : J \mapsto 2^M \setminus \emptyset$ is a *ν-partitioning assignment* if for every pair of messages j, j', either $\phi(j) = \phi(j')$ or $\phi(j) \cap \phi(j') = \emptyset$, and ϕ assigns the messages to ν different link subsets.

Computing an appropriate fault-tolerant collection of disjoint link subsets is an interesting optimization problem by itself. A feasible solution $\mathcal{M}$ satisfies the constraint that $\mathrm{Rel}(\mathcal{M}) \geq 1 - \epsilon$. For identical capacity links, the most natural objective is to maximize the number of subsets in $\mathcal{M}$ (equivalently, the number of reliable links used by the load balancing algorithm). For arbitrary capacities, this objective generalizes to maximizing the total effective capacity of $\mathcal{M}$.

Problem 2 (Maximum Fault-Tolerant Partition)
INPUT: *A set of faulty parallel links $M = \{(c_1, f_1), \ldots, (c_m, f_m)\}$, and a rational number $\epsilon \in (0, 1)$.*
OUTPUT: *A collection $\mathcal{M} = \{M_1, \ldots, M_\nu\}$ of disjoint subsets of M with $Rel(\mathcal{M}) \geq 1 - \epsilon$ that maximizes $\sum_{\ell=1}^{\nu} c(M_\ell)$.*

The problem of Maximum Fault-Tolerant Partition is **NP**-hard. More precisely, given m identical capacity links with rational failure probabilities and a rational number $\epsilon \in (0, 1)$, it is **NP**-complete to decide whether the links can be partitioned into sets M_1 and M_2 with $\mathrm{Rel}(M_1) \cdot \mathrm{Rel}(M_2) \geq 1 - \epsilon$.

Key Results

Theorem 1 *There is a 2-approximation algorithm for Maximum Fault-Tolerant Partition of identical capacity links. The time complexity of the algorithm is $O((m - \sum_{i \in M} \ln f_i) \ln m)$.*

Theorem 2 *For every constant $\delta > 0$, there is a $(8 + \delta)$-approximation algorithm for Maximum Fault-Tolerant Partition of capacitated links. The time complexity of the algorithm is polynomial in the input size and $1/\delta$.*

To demonstrate the efficiency of the partitioning approach for Maximum Fault-Tolerant Congestion, Fotakis and Spirakis prove that for certain instances, the reliability of the most reliable partitioning assignment bounds from above the reliability of any other assignment with the same congestion (see Fig. 1 for an example).

Lemma 3 *For any positive integers Λ, ν, μ and any rational $f \in (0, 1)$, let ϕ be a redundant assignment of $\Lambda\nu$ unit size messages to $\nu\mu$ identical capacity links with failure probability f. Let ϕ_ν be the ν-partitioning assignment that assigns Λ messages to each of ν disjoint subsets consisting of μ links each. If $Cong(\phi) \leq \Lambda = Cong(\phi_\nu)$, then $Rel(\phi) \leq (1 - f^\mu)^\nu = Rel(\phi_\nu)$.*

Based on the previous upper bound on the reliability of any redundant assignment, [1] presents polynomial-time approximation algorithms for Maximum Fault-Tolerant Congestion.

Theorem 4 *There is a quasi-linear-time 4-approximation algorithm for Maximum Fault-Tolerant Congestion on identical capacity links.*

Theorem 5 *There is a polynomial-time* $2\lceil \ln(m/\epsilon)/\ln(1/f_{\max})\rceil$*-approximation algorithm for Maximum Fault-Tolerant Congestion on instances with unit size messages and capacitated links.*

Applications

In many applications dealing with faulty components (e. g. fault-tolerant network design, fault-tolerant routing), a combinatorial structure (e. g. a graph, a hypergraph) should optimally tolerate random faults with respect to a given property (e. g. connectivity, non-existence of isolated points). For instance, Lomonosov [5] derived tight upper and lower bounds on the probability that a graph remains connected under random edge faults. Using the bounds of Lomonosov, Karger [3] obtained improved theoretical and practical results for the problem of estimating the reliability of a graph. In this work, Lemma 3 provides a tight upper bound on the probability that isolated nodes do not appear in a not necessarily connected hypergraph with $\Lambda\nu$ nodes and $\nu\mu$ "faulty" hyperedges of cardinality Λ.

More precisely, let ϕ be any assignment of $\Lambda\nu$ unit size messages to $\nu\mu$ identical links that assigns every message to μ links and Λ messages to every link. Then ϕ corresponds to a hypergraph H_ϕ, where the set of nodes consists of $\Lambda\nu$ elements corresponding to the unit size messages and the set of hyperedges consists of $\nu\mu$ elements corresponding to the identical links. Every hyperedge contains the messages assigned to the corresponding link and has cardinality Λ (see Fig. 1 for a simple example with $\Lambda = 2$, $\nu = 2$, and $\mu = 4$). Clearly, an assignment ϕ is feasible for a set of links $M' \subseteq M$ iff the removal of the hyperedges corresponding to the links in $M \setminus M'$ does not leave any isolated nodes[3] in H_ϕ. Lemma 3 implies that the hypergraph corresponding to the most reliable ν-partitioning assignment maximizes the probability that isolated nodes do not appear when hyperedges are removed equiprobably and independently.

The previous work on fault-tolerant network design and routing mostly focuses on the worst-case fault model, where a feasible solution must tolerate any configuration of a given number of faults. The work of Gasieniec et al. [2] studies the fault-tolerant version of minimizing congestion of virtual path layouts in a complete ATM network. In addition to several results for the worst-case fault model, [2] constructs a virtual path layout of logarithmic congestion that tolerates random faults with high probability. On the other hand, the work of Fotakis and Spirakis shows how to construct redundant assignments that tolerate random faults with a probability given as part of the input and achieve a congestion close to optimal.

[3]For a node v, let $\deg_H(v) \equiv |\{e \in E(H) : v \in e\}|$. A node v is isolated in H if $\deg_H(v) = 0$.

Open Problems

An interesting research direction is to determine the computational complexity of Minimum Fault-Tolerant Congestion and related problems. The decision version of Minimum Fault-Tolerant Congestion is included in the class of languages decided by a polynomial-time non-deterministic Turing machine that reduces the language to a single call of a **#P** oracle. After calling the oracle once, the Turing machine rejects if the oracle's outcome is less than a given threshold and accepts otherwise. This class is denoted $\mathbf{NP}^{\#\mathbf{P}[1,\mathrm{comp}]}$ in [1]. In addition to Minimum Fault-Tolerant Congestion, $\mathbf{NP}^{\#\mathbf{P}[1,\mathrm{comp}]}$ includes the decision version of Stochastic Knapsack considered in [4]. A result of Toda and Watanabe [6] implies that $\mathbf{NP}^{\#\mathbf{P}[1,\mathrm{comp}]}$ contains the entire Polynomial Hierarchy. A challenging open problem is to determine whether the decision version of Minimum Fault-Tolerant Congestion is complete for $\mathbf{NP}^{\#\mathbf{P}[1,\mathrm{comp}]}$.

A second direction for further research is to consider the generalizations of other fundamental optimization problems (e. g. shortest paths, minimum connected subgraph) under random faults. In the fault-tolerant version of minimum connected subgraph for example, the input consists of a graph whose edges fail independently with given probabilities, and a rational number $\epsilon \in (0, 1)$. The goal is to compute a spanning subgraph with a minimum number of edges whose reliability is at least $1 - \epsilon$.

Cross References

- ► Approximation Schemes for Bin Packing
- ► Bin Packing
- ► Connectivity and Fault-Tolerance in Random Regular Graphs
- ► List Scheduling

Recommended Reading

1. Fotakis, D., Spirakis, P.: Minimum Congestion Redundant Assignments to Tolerate Random Faults. Algorithmica **32**, 396–422 (2002)
2. Gasieniec, L., Kranakis, E., Krizanc, D., Pelc, A.: Minimizing Congestion of Layouts for ATM Networks with Faulty Links. In: Penczek, W., Szalas, A. (eds.) Proceedings of the 21st International Symposium on Mathematical Foundations of Computer Science. Lecture Notes in Computer Science, vol. 1113, pp. 372–381. Springer, Berlin (1996)

3. Karger, D.: A Randomized Fully Polynomial Time Approximation Scheme for the All-Terminal Network Reliability Problem. SIAM J. Comput. **29**, 492–514 (1999)
4. Kleinberg, J., Rabani, Y., Tardos, E.: Allocating Bandwidth for Bursty Connections. SIAM J. Comput. **30**, 191–217 (2000)
5. Lomonosov, M.: Bernoulli Scheme with Closure. Probl. Inf. Transm. **10**, 73–81 (1974)
6. Toda, S., Watanabe, O.: Polynomial-Time 1-Turing Reductions from #**PH** to #**P**. Theor. Comput. Sci. **100**, 205–221 (1992)

Minimum Energy Broadcasting in Wireless Geometric Networks
2005; Ambühl

Christoph Ambühl
Department of Computer Science,
University of Liverpool, Liverpool, UK

Keywords and Synonyms

Energy-efficient broadcasting in wireless networks

Problem Definition

In the most common model for wireless networks, stations are represented by points in $\mathbb{R}^d$. They are equipped with a omnidirectional transmitter and receiver which enables them to communicate with other stations. In order to send a message from a station s to a station t, station s needs to emit the message with enough power such that t can receive it. It is usually assumed that the power required by a station s to transmit data directly to station t is $\|st\|^\alpha$, for some constant $\alpha \geq 1$, where $\|st\|$ denotes the distance between s and t.

Because of the omnidirectional nature of the transmitters and receivers, a message sent by a station s with power r^α can be received by all stations within a disc of radius r around s. Hence the energy required to send a message from a station s directly to a set of stations S' is determined by $\max_{v \in S'} \|sv\|^\alpha$.

An instance of the *minimum energy broadcast routing problem in wireless networks* (MEBR) consists of a set of stations S and a constant $\alpha \geq 1$. One of the stations in S is designated as the source station s_0. The goal is to send a message at minimum energy cost from s_0 to all other stations in S. This operation is called *broadcast*.

In the case $\alpha = 1$, the optimal solution is to send the message directly from s_0 to all other stations. For $\alpha > 1$, sending the message via intermediate stations which forward it to other stations is often more energy efficient.

A solution of the MEBR instance can be described in terms of a so-called *broadcast tree*. That is, a directed spanning tree of S which contains directed paths from s_0 to all other vertices. The solution described by a broadcast tree T is the one in which every station forwards the message to all its out-neighbors in T.

Problem 1 (MEBR)
INSTANCE: *A set S of points in R^d, $s_0 \in S$ designated as the source, and a constant α.*
SOLUTION: *A broadcast tree T of S.*
MEASURE: *The objective is to minimize the total energy needed to broadcast a message from s_0 to all other nodes, which can be expressed by*

$$\sum_{u \in S} \max_{v \in \delta(u)} \|uv\|^\alpha \,, \tag{1}$$

where $\delta(u)$ denotes the set of out-neighbors of station u in T.

The MEBR problem is known to be NP-hard for $d \geq 2$ and $\alpha > 1$ [2]. APX-hardness is known for $d \geq 3$ [5].

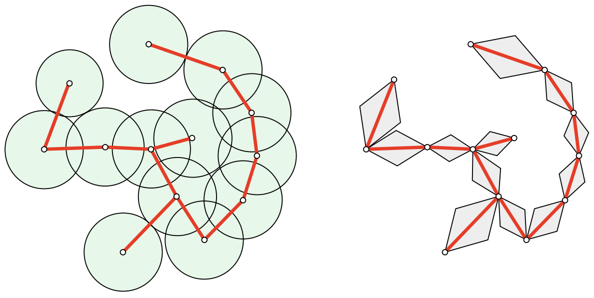
Key Results

Numerous heuristics have been proposed for this problem. Only a few of them have been analyzed theoretically. The one which attains the best approximation guarantee is the so-called MST-heuristic [12].

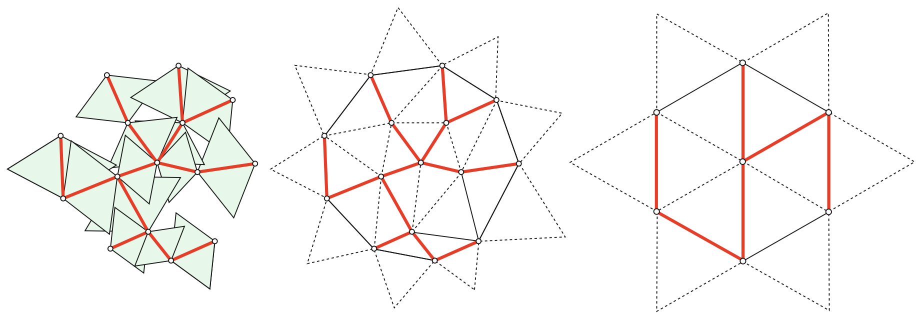
MST-HEURISTIC: Compute a minimum spanning tree of S (*mst(S)*) and turn it into an broadcast tree by directing the edges.

Theorem 1 [1] *In the Euclidean plane, the MST-heuristic is a 6 approximation algorithm for MEBR for all $\alpha \geq 2$.*

Theorem 2 [9] *In the Euclidean three-dimensional space, the MST-heuristic is a 18.8 approximation algorithm for MEBR for all $\alpha \geq 3$.*



Minimum Energy Broadcasting in Wireless Geometric Networks, Figure 1
Illustration of the first and second approach for bounding *w(S)*. In both approaches, *w(S)* is bounded in terms of the total area covered by the shapes



Minimum Energy Broadcasting in Wireless Geometric Networks, Figure 2
Illustration of the tight bound for $d = 2$. The total area of the equilateral triangles on the left is bounded by its extended convex hull shown in the middle. The point set that maximizes area of the extended convex hull is the star shown on the right

For $\alpha < d$, the MST-heuristic does not provide a constant approximation ratio. The d-dimensional kissing numbers represent lower bounds for the performance of the MST-heuristic. Hence the analysis for $d = 2$ is tight, whereas for $d = 3$ the lower bound is 12.

Analysis

The analysis of the MST-heuristic is based on good upper bounds for

$$w(S) := \sum_{e \in mst(S)} \|e\|^{\alpha} , \qquad (2)$$

which obviously is an upper bound on (1). The radius of an instance of MEBR is the distance between s_0 to the station furthest from s_0. It turns out that the MST-heuristic performs worst on instances of radius 1 whose optimal solution is to broadcast the message directly from s_0 to all other stations. Since the optimal value for such instances is 1, the approximation ratio follows from good upper bounds on *w(S)* for instances with radius 1.

The rest of this section focuses on the case $d = \alpha = 2$. There are two main approaches for upper bounding *w(S)*. In both approaches, *w(S)* is upper bounded in terms of the area of particular kinds of shapes associated with either the stations or with the edges of the MST.

In the first approach, the shapes are disks of radius $m/2$ placed around every station of S, where m is the length of the longest edge of *mst(S)*. Let A denote the area covered by the disks. One can prove $w(S) \leq \frac{4}{\pi}\left(A - \pi(m/2)^2\right)$. Assuming that S has radius 1, one can prove $w(S) \leq 8$ quite easily [4]. This approach can even be extended to obtain $w(S) \leq 6.33$ [8], and it can be generalized for $d \geq 2$.

In the second approach [7,11], *w(S)* is expressed in terms of shapes associated with the edges of *mst(S)*, e. g., diamond shapes as shown on the right of Fig. 1. The area of a diamond for an edge e is equal to $\|e\|^2/(2\sqrt{3})$. Since one can prove that the diamonds never intersect, one obtains $w(S) = A/(2\sqrt{3})$. For instances with radius 1, one can get $w(S) \leq 12.15$.

For the 2-dimensional case, one can even obtain a matching upper bound [1]. The shapes used in this proof are equilateral triangles, arranged in pairs along every edge of the MST. As can be seen on the left of Fig. 2, these shapes do intersect. Still one can obtain a good upper bound on their total area by means of the convex hull of S:

Let the *extended convex hull of S* be the convex hull of S extended by equilateral triangles along the border of the convex hull. One can prove that the total area generated by the equilateral triangle shapes along the edges of *mst(S)* is upper bounded by the area of the extended convex hull of S. By showing that for instances of radius 1 the area of the extended convex hull is maximized by the point configuration shown on the right of Fig. 2, the matching upper bound of 6 can be established.

Applications

The MEBR problem is a special case of a large class of problems called *range assignment problems*. In all these problems, the goal is to assign a range to each station such that a certain type of communication operation such as broadcast, all-to-1 (gathering), all-to-all (gossiping), can be accomplished. See [3] for a survey on range assignment problems.

It is worth noting that the problem of upper bounding *w(S)* has already been considered in different contexts. The idea of using diamond shapes to upper bound the length of an MST has already been used by Gilbert and Pollak in [6]. Steele [10] makes use of space filling curves to bound *w(S)*.

Open Problems

An obvious open problem is to close the gap in the analysis of the MST-heuristic for the three dimensional case. This might be very difficult, as the lower bound from the kissing number might not be tight.

Even in the plane, the approximation ratio of the MST-heuristic is quite large. It would be interesting to see a different approach for the problem, maybe based on LP-rounding. It is still not known whether MEBR is APX-hard for instances in the Euclidean plane. Hence there might exist a PTAS for this problem.

Cross References

▶ Broadcasting in Geometric Radio Networks
▶ Deterministic Broadcasting in Radio Networks
▶ Geometric Spanners
▶ Minimum Geometric Spanning Trees
▶ Minimum Spanning Trees
▶ Randomized Broadcasting in Radio Networks
▶ Randomized Gossiping in Radio Networks

Recommended Reading

1. C. Ambühl: An optimal bound for the MST algorithm to compute energy efficient broadcast trees in wireless networks. In: Proceedings of 32th International Colloquium on Automata, Languages and Programming (ICALP). Lecture Notes in Computer Science, vol. 3580, pp. 1139–1150. Springer, Berlin (2005)
2. Clementi, A., Crescenzi, P., Penna, P., Rossi, G., Vocca, P.: On the Complexity of Computing Minimum Energy Consumption Broadcast Subgraphs. In: Proceedings of the 18th Annual Symposium on Theoretical Aspects of Computer Science (STACS), pp. 121–131 (2001)
3. Clementi, A., Huiban, G., Penna, P., Rossi, G., Verhoeven, Y.: Some Recent Theoretical Advances and Open Questions on Energy Consumption in Ad-Hoc Wireless Networks. In: Proceedings of the 3rd Workshop on Approximation and Randomization Algorithms in Communication Networks (ARACNE), pp. 23–38 (2002)
4. Flammini, M., Klasing, R., Navarra, A., Perennes, S.: Improved approximation results for the minimum energy broadcasting problem. In: Proceedings of the 2004 joint workshop on Foundations of mobile computing (2004)
5. Fuchs, B.: On the hardness of range assignment problems. In: Proceedings of the 6th Italian Conference on Algorithms and Complexity (CIAC), pp. 127–138 (2006)
6. Gilbert, E.N., Pollak, H.O.: Steiner minimal trees. SIAM J. Appl. Math. **16**, 1–29 (1968)
7. Klasing, R., Navarra, A., Papadopoulos, A., Perennes, S.: Adaptive broadcast consumption (ABC), a new heuristic and new bounds for the minimum energy broadcast routing problem. In: Proceeding of the 3rd IFIP-TC6 international networking conference (NETWORKING), pp. 866–877 (2004)
8. Navarra, A.: Tighter bounds for the minimum energy broadcasting problem. In: Proceedings of the 3rd International Symposium on Modeling and Optimization in Mobile, Ad-hoc and Wireless Networks (WiOpt), pp. 313–322 (2005)
9. Navarra, A.: 3-d minimum energy broadcasting. In: Proceedings of the 13th Colloquium on Structural Information and Communication Complexity (SIROCCO), pp. 240–252 (2006)
10. Steele, J.M.: Cost of sequential connection for points in space. Oper. Res. Lett. **8**, 137–142 (1989)
11. Wan, P.-J., Calinescu, G., Li, X.-Y., Frieder, O.: Minimum-energy broadcasting in static ad hoc wireless networks. Wirel. Netw. **8**, 607–617 (2002)
12. Wieselthier, J.E., Nguyen, G.D., Ephremides, A.: Energy-efficient broadcast and multicast trees in wireless networks. Mobile Netw. Appl. **7**, 481–492 (2002)

Minimum Energy Cost Broadcasting in Wireless Networks

2001; Wan, Calinescu, Li, Frieder

PENG-JUN WAN, XIANG-YANG LI, OPHIR FRIEDER
Department of Computer Science, Illinois Institute of Technology, Chicago, IL, USA

Keywords and Synonyms

Minimum energy broadcast; MST; MEB

Problem Definition

Ad hoc wireless networks have received significant attention in recent years due to their potential applications in battlefield, emergency disaster relief and other applications [11,15]. Unlike wired networks or cellular networks, no wired backbone infrastructure is installed in ad hoc wireless networks. A communication session is achieved either through a single-hop transmission if the communication parties are close enough, or through relaying by intermediate nodes otherwise. Omni-directional antennas are used by all nodes to transmit and receive signals. They are attractive in their broadcast nature. A single transmission by a node can be received by many nodes within its vicinity. This feature is extremely useful for multicasting/broadcasting communications. For the purpose of energy conservation, each node can dynamically adjust its transmitting power based on the distance to the receiving node and the background noise. In the most common power-attenuation model [10], the signal power falls as $\frac{1}{r^\kappa}$, where r is the distance from the transmitter antenna and κ is a real *constant* between 2 and 4 dependent on the wireless environment. Assume that all receivers have the same power threshold for signal detection, which is typically normalized to one. With these assumptions, the power required to support a link between two nodes separated by a distance r is r^κ. A key observation here is that relaying a signal between two nodes may result in lower total transmission power than communicating over a large distance due to the nonlinear power attenuation. They assume the network nodes are given as a finite point[1] set P in a two-dimensional plane. For any real number κ, they use $G^{(\kappa)}$ to denote the weighted complete graph over P in which the weight of an edge e is $\|\mathbf{e}\|^{\kappa}$.

The minimum-energy unicast routing is essentially a shortest-path problem in $G^{(\kappa)}$. Consider any unicast path from a node $\mathbf{p} = \mathbf{p}_0 \in P$ to another node $\mathbf{q} = \mathbf{p}_m \in P$: $\mathbf{p}_0\mathbf{p}_1 \cdots \mathbf{p}_{m-1}\mathbf{p}_m$. In this path, the transmission power of each node $\mathbf{p}_i$, $0 \le i \le m-1$, is $\|\mathbf{p}_i\mathbf{p}_{i+1}\|^{\kappa}$ and the transmission power of $\mathbf{p}_m$ is zero. Thus the total transmission energy required by this path is $\sum_{i=0}^{m-1} \|\mathbf{p}_i\mathbf{p}_{i+1}\|^{\kappa}$, which is the total weight of this path in G^{κ}. So by applying any shortest-path algorithm such as the Dijkstra's algorithm [5], one can solve the minimum-energy unicast routing problem.

However, for broadcast applications (in general multicast applications), Minimum-Energy Routing is far more challenging. Any broadcast routing is viewed as an arborescence (a directed tree) T, rooted at the source node of the broadcasting, that spans all nodes. Use $f_T(\mathbf{p})$ to denote the transmission power of the node $\mathbf{p}$ required by T. For any leaf node $\mathbf{p}$ of T, $f_T(\mathbf{p}) = 0$. For any internal node $\mathbf{p}$ of T,

$$f_T(\mathbf{p}) = \max_{\mathbf{pq} \in T} \|\mathbf{pq}\|^{\kappa} ,$$

in other words, the κ-th power of the longest distance between $\mathbf{p}$ and its children in T. The total energy required by T is $\sum_{\mathbf{p}\in P} f_T(\mathbf{p})$. Thus the minimum-energy broadcast routing problem is different from the conventional link-based minimum spanning tree (MST) problem. Indeed, while the MST can be solved in polynomial time by algorithms such as Prim's algorithm and Kruskal's algorithm [5], it is NP-hard [4] to find the minimum-energy broadcast routing tree for nodes placed in two-dimensional plane. In its general graph version, the minimum-energy broadcast routing can also be shown to be NP-hard [7], and even worse, it can not be approximated within a factor of $(1-\epsilon)\log \Delta$, unless $NP \subseteq DTIME\left[n^{O(\log\log n)}\right]$, by an approximation-preserving reduction from the Connected Dominating Set problem [8], where Δ is the maximal degree and ϵ is any arbitrary small positive constant.

Three greedy heuristics have been proposed for the minimum-energy broadcast routing problem by [15]. The MST heuristic first applies the Prim's algorithm to obtain a MST, and then orient it as an arborescence rooted at the source node. The SPT heuristic applies the Dijkstra's algorithm to obtain a SPT rooted at the source node. The BIP heuristic is the node version of Dijkstra's algorithm for SPT. It maintains, throughout its execution, a single arborescence rooted at the source node. The arborescence starts from the source node, and new nodes are added to the arborescence one at a time on the minimum incremental cost basis until all nodes are included in the arborescence. The incremental cost of adding a new node to the arborescence is the minimum additional power increased by some node in the current arborescence to reach this new node. The implementation of BIP is based on the standard Dijkstra's algorithm, with one fundamental difference on the operation whenever a new node q is added. Whereas the Dijkstra's algorithm updates the node weights (representing the current knowing distances to the source node), BIP updates the cost of each link (representing the incremental power to reach the head node of the directed link). This update is performed by subtracting the cost of the added link pq from the cost of every link qr that starts from q to a node r not in the new arborescence.

[1]The terms node, point and vertex are interchangeable here: node is a network term, point is a geometric term, and vertex is a graph term.

Key Results

The performance of these three greedy heuristics have been evaluated in [15] by simulation studies. However, their analytic performances in terms of the approximation ratio remained open until [13]. The work of Wan et al. [13] derived the bounds on their approximation ratios.

Let us begin with the SPT algorithm. Let ϵ be a sufficiently small positive number. Consider m nodes $\mathbf{p}_1, \mathbf{p}_2, \cdots, \mathbf{p}_m$ evenly distributed on a cycle of radius 1 centered at a node $\mathbf{o}$. For $1 \le i \le m$, let $\mathbf{q}_i$ be the point in the line segment $\mathbf{op}_i$ with $\left\|\mathbf{oq}_i\right\| = \epsilon$. They consider a broadcasting from the node $\mathbf{o}$ to these $n = 2m$ nodes $\mathbf{p}_1, \mathbf{p}_2, \cdots, \mathbf{p}_m, \mathbf{q}_1, \mathbf{q}_2, \cdots, \mathbf{q}_m$. The SPT is the superposition of paths $\mathbf{oq}_i\mathbf{p}_i$, $1 \le i \le m$. Its total energy consumption is $\epsilon^2 + m(1-\epsilon)^2$. On the other hand, if the transmission power of node $\mathbf{o}$ is set to 1, then the signal can reach all other points. Thus the minimum energy consumed by all broadcasting methods is at most 1. So the approximation ratio of SPT is at least $\epsilon^2 + m(1-\epsilon)^2$. As $\epsilon \longrightarrow 0$, this ratio converges to $\frac{n}{2} = m$.

They [13] also proved that

Theorem 1 *The approximation ratio of MST is at least 6 for any $\kappa \ge 2$.*

Theorem 2 *The approximation ratio of BIP is at least $\frac{13}{3}$ for any $\kappa = 2$.*

They then derived the upper bounds by extensively using the geometric structures of Euclidean MSTs (EMST). They first observed that as long as the cost of a link is an increasing function of the Euclidean length of the link, the set of MSTs of any point set *coincides* with the set of Euclidean MSTs of the same point set. They proved a key result about an upper bound on the parameter $\sum_{e \in MST(P)} \|e\|^2$ for any finite point set P inside a disk with radius one.

Theorem 3 *Let c be the supreme of $\sum_{e \in MST(P)} \|e\|^2$ over all such point sets P. Then $6 \le c \le 12$.*

The following lemma proved in [13] is used to bound the energy cost for broadcast when each node can dynamically adjust its power.

Lemma 4 *For any point set P in the plane, the total energy required by any broadcasting among P is at least $\frac{1}{c} \sum_{e \in MST(P)} \|e\|^{\kappa}$.*

Lemma 5 *For any broadcasting among a point set P in a two-dimensional plane, the total energy required by the arborescence generated by the BIP algorithm is at most $\sum_{e \in MST(P)} \|e\|^{\kappa}$.*

Thus, they conclude the following two theorems.

Theorem 6 *The approximation ratio of EMST is at most c, and therefore is at most 12.*

Theorem 7 *The approximation ratio of BIP is at most c, and therefore is at most 12.*

Later, Wan et al. [14] studied the energy efficient multicast for wireless networks when each node can dynamically adjust its power. Given a set of receivers Q, the problem Min-Power Asymmetric Multicast seeks, for any given communication session, an arborescence T of minimum total power which is rooted at the source node s and reaches all nodes in Q. As a generalization of Min-Power Asymmetric Broadcast Routing, Min-Power Asymmetric Multicast Routing is also NP-hard. Wieselthier et al. [15] adapted their three broadcasting heuristics to three multicasting heuristics by a technique of pruning, which was called as pruned minimum spanning tree (P-MST), pruned shortest-path tree (P-SPT), and pruned broadcasting incremental power (P-BIP), respectively in [14]. The idea is as follows. They first obtain a spanning tree rooted at the source of a given multicast session by applying any of the three broadcasting heuristics. They then eliminate from the spanning arborescence all nodes which do not have any descendant in Q. They [14] show by constructing examples that all structures P-SPT, P-MST and P-BIP could have approximation ratio as large as $\Theta(n)$ in the worst case for multicast. They then further proposed a multicast scheme with a constant approximation ratio on the total energy consumption. Their protocol for Min-Power Asymmetric Multicast Routing is based on Takahashi-Matsuyama Steiner tree heuristic [12]. Initially, the multicast tree T contains only the source node. At each iterative step, the multicast tree T is grown by one path from some node in T to some destination node from Q that is not yet in the tree T. The path must have the least total power among all such paths from a node in T to a node in $Q - T$. This procedure is repeated until all required nodes are included in T. This heuristic is referred to as Shortest Path First (SPF).

Theorem 8 *For asymmetric multicast communication, the approximation ratio of SPF is between 6 and $2c$, which is at most 24.*

Applications

Broadcasting and multicasting in wireless ad hoc networks are critical mechanisms in various applications such as information diffusion, wireless networks, and also for maintaining consistent global network information. Broadcasting is often necessary in MANET routing protocols. For example, many unicast routing protocols such as Dynamic Source Routing (DSR), Ad Hoc On Demand Distance Vector (AODV), Zone Routing Protocol (ZRP), and Location Aided Routing (LAR) use broadcasting or a derivation of it to establish routes. Currently, these protocols all rely on a simplistic form of broadcasting called *flooding*, in which each node (or all nodes in a localized area) retransmits each received unique packet exactly one time. The main problems with flooding are that it typically causes unproductive and often harmful bandwidth congestion, as well as inefficient use of node resources. Broadcasting is also more efficient than sending multiple copies the same packet through unicast. It is highly important to use power-efficient broadcast algorithms for such networks since wireless devises are often powered by batteries only.

Open Problems

There are some interesting questions left for further study. For example, the exact value of the constant c remains unsolved. A tighter upper bound on c can lead to tighter upper bounds on the approximation ratios of both the link-based MST heuristic and the BIP heuristic. They conjecture that the exact value for c is 6, which seems to be true based on their extensive simulations. The second question is what is the approximation lower bound for minimum energy broadcast? Is there a PTAS for this problem?

So far, all the known theoretically good algorithms either assume that the power needed to support a link uv is proportional to $\|uv\|^{\kappa}$ or is a fixed cost that is independent of the neighboring nodes that it will communicate with. In practice, the energy consumption of a node is neither solely dependent on the distance to its farthest neighbor, nor totally independent of its communication neighbor. For example, a more general power consumption model for a node u would be $c_1 + c_2 \cdot \|uv\|^\kappa$ for some constants $c_1 \geq 0$ and $c_2 \geq 0$ where v is its farthest communication neighbor in a broadcast structure. No theoretical result is known about the approximation of the optimum broadcast or multicast structure under this model. When $c_2 = 0$, this is the case where all nodes have a fixed power for communication. Minimizing the total power used by a reliable broadcast tree is equivalent to the minimum connected dominating set problem (MCDS), i. e., minimize the number of nodes that relay the message, since all relaying nodes of a reliable broadcast form a connected dominating set (CDS). Notice that recently a PTAS [2] has been proposed for MCDS in UDG graph.

Another important question is how to find efficient broadcast/multicast structures such that the delay from the source node to the last node receiving message is bounded by a predetermined value while the total energy consumption is minimized. Notice that here the delay of a broadcast/multicast based on a tree is not simply the height of the tree: many nodes cannot transmit simultaneously due to the interference.

Cross References

► Broadcasting in Geometric Radio Networks
► Deterministic Broadcasting in Radio Networks

Recommended Reading

1. Ambühl, C.: An Optimal Bound for the MST Algorithm to Compute Energy Efficient Broadcast Trees in Wireless Networks. In: Proceedings of 32th International Colloquium on Automata, Languages and Programming (ICALP). LNCS, vol. 3580, pp. 1139–1150 (2005)
2. Cheng, X., Huang, X., Li, D., Du, D.-Z.: Polynomial-time approximation scheme for minimum connected dominating set in ad hoc wireless networks. Netw. **42**, 202–208 (2003)
3. Chvátal, V.: A Greedy Heuristic for the Set-Covering Problem. Math. Oper. Res. **4**(3), 233–235 (1979)
4. Clementi, A., Crescenzi, P., Penna, P., Rossi, G., Vocca, P.: On the complexity of computing minimum energy consumption broadcast subgraphs. In: 18th Annual Symposium on Theoretical Aspects of Computer Science. LNCS, vol. 2010, pp. 121–131 (2001)
5. Cormen, T.J., Leiserson, C.E., Rivest, R.L.: Introduction to Algorithms. MIT Press and McGraw-Hill, Columbus (1990)
6. Flammini, M., Navarra, A., Klasing, R., Pérennes, A.: Improved approximation results for the minimum energy broadcasting problem. DIALM-POMC, pp. 85–91. ACM Press, New York (2004)
7. Garey, M.R., Johnson, D.S.: Computers and Intractability: a Guide to the Theory of NP-Completeness. W.H. Freeman and Company, New York (1979)
8. Guha, S., Khuller, S.: Approximation Algorithms for Connected Dominating Sets. Algorithmica **20**, 347–387 (1998)
9. Preparata, F.P., Shamos, M.I.: Computational Geometry: an Introduction. Springer, New York (1985)
10. Rappaport, T.S.: Wireless Communications: Principles and Practices. Prentice Hall, IEEE Press, Piscataway (1996)
11. Singh, S., Raghavendra, C.S., Stepanek, J.: Power-Aware Broadcasting in Mobile Ad Hoc Networks. In: Proceedings of IEEE PIMRC'99, Osaka, September 1999
12. Takahashi, H., Matsuyama, A.: An approximate solution for steiner problem in graphs. Mathematica Japonica **24**(6), 573–577 (1980)
13. Wan, P.-J., Calinescu, G., Li, X.-Y., Frieder, O.: Minimum-energy broadcast routing in static ad hoc wireless networks. ACM Wirel. Netw. Preliminary version appeared in IEEE INFOCOM (2000)**8**(6), 607–617 (2002)
14. Wan, P.-J., Calinescu, G., Yi, C.-W.: Minimum-power multicast routing in static ad hoc wireless networks. IEEE/ACM Trans. Netw. **12**, 507–514 (2004)
15. Wieselthier, J.E., Nguyen, G.D., Ephremides, A.: On the Construction of energy-Efficient Broadcast and Multicast Trees in Wireless Networks. IEEE Infocom **2**, 585–594 (2000)

Minimum Flow Time

1997; Leonardi, Raz

NIKHIL BANSAL
IBM Research, IBM, Yorktown Heights, NY, USA

Keywords and Synonyms

Response time; Sojourn time

Problem Definition

The problem is concerned with efficiently scheduling jobs on a system with multiple resources to provide a good quality of service. In scheduling literature, several models have been considered to model the problem setting and several different measures of quality have been studied. This note considers the following model: There are several identical machines, and jobs are released over time. Each job is characterized by its size, which is the amount of processing it must receive to be completed, and its release time, before which it cannot be scheduled. In this model, Leonardi and Raz studied the objective of minimizing the average flow time of the jobs, where the flow time of a job is duration of time since it is released until its processing requirement is met. Flow time is also referred to as response time or sojourn time and is a very natural and commonly used measure of the quality of a schedule.

Notations Let $\mathcal{J} = \{1, 2, \ldots, n\}$ denote the set of jobs in the input instance. Each job j is characterized by its release time r_j and its processing requirement p_j. There is a collection of m identical machines, each having the same processing capability. A schedule specifies which job executes at what time on each machine. Given a schedule, the completion time c_j of a job is the earliest time at which job j receives p_j amount of service. The flow time f_j of j is defined as $c_j - r_j$. A schedule is said to be preemptive, if a job can be interrupted arbitrarily, and its execution can be resumed later from the point of interruption without any penalty. A schedule is non-preemptive if a job cannot be interrupted once it is started. In the context of multiple machines, a schedule is said to be migratory, if a job can be moved from one machine to another during its execution without any penalty. In the offline model, all the jobs J are given in advance. In scheduling algorithms, the online model is usually more realistic than the offline model.

Key Results

For a single machine, it is a folklore result that the Shortest Remaining Processing Time (SRPT) policy, that at any time works on the job with the least remaining processing time is optimal for minimizing the average flow time. Note that SRPT is an online algorithm, and is a preemptive scheduling policy.

If no preemption is allowed Kellerer, Tautenhahn, and Woeginger [6] gave an $O(n^{1/2})$ approximation algorithm for minimizing the flow time on a single machine, and also showed that no polynomial time algorithm can have an approximation ratio of $n^{1/2-\varepsilon}$ for any $\varepsilon > 0$ unless P=NP.

Leonardi and Raz [8] gave the first non-trivial results for minimizing the average flow time on multiple machines. Later, a simpler presentation of this result was given by Leonardi [7]. The main result of [8] is the following.

Theorem 1 ([8]) *On multiple machines, the SRPT algorithm is* $O(\min(\log(n/m), \log P))$ *competitive for minimizing average flow time, where P is the maximum to minimum job size ratio.*

They also gave a matching lower bound (up to constant factors) on the competitive ratio.

Theorem 2 ([8]) *For the problem of minimizing flow time on multiple machines, any online algorithm has a competitive ratio of* $\Omega(\min(\log(n/m), \log P))$*, even when randomization is allowed.*

Note that minimizing the average flow time is equivalent to minimizing the total flow time. Suppose each job pays $1 at each time unit it is alive (i. e. unfinished), then the total payment received is equal to the total flow time. Summing up the payment over each time step, the total flow time can be expressed as the summation over the number of unfinished jobs at each time unit. As SRPT works on jobs that can be finished as soon as possible, it seems intuitively that it should have the least number of unfinished jobs at any time. While this is true for a single machine, it is not true for multiple machines (as shown in an example below). The main idea of [8] was to show that at any time, the number of unfinished jobs under SRPT is "essentially" no more than $O(\min(\log P))$ times that under any other algorithm. To do this, they developed a technique of grouping jobs into a logarithmic number of classes according to their remaining sizes and arguing about the total unfinished work in these classes. This technique has found a lot of uses since then to obtain other results. To obtain a guarantee in terms of n, some additional ideas are required.

The instance below shows how SRPT could deviate from optimum in the case of multiple machines. This instance is also the key component in the lower bound construction in Theorem 2 above. Suppose there are two machines, and three jobs of size 1, 1, and 2 arrive at time $t = 0$. SRPT would schedule the two jobs of size 1 at $t = 0$ and then work on size 2 job at time $t = 1$. Thus, it has one unit of unfinished work at $t = 2$. However, the optimum could schedule the size 2 job at time 0, and finish all these jobs by time 2. Now, at time $t = 2$ three more jobs with sizes 1/2, 1/2, and 1 arrive. Again, SRPT will work on size 1/2 jobs first, and it can be seen that it will have two unfinished jobs with remaining work 1/2 each at $t = 3$, whereas the optimum can finish all these jobs by time 3. This pattern is continued by giving three jobs of size 1/4, 1/4, and 1/2 at $t = 3$ and so on. After k steps, SRPT will have k jobs with sizes $1/2, 1/4, 1/8, \ldots, 1/2^{k-2}, 1/2^{k-1}, 1/2^{k-1}$, while the optimum has no jobs remaining. Now the adversary can give 2 jobs of size $1/2^k$ each every $1/2^k$ time units for a long time, which implies that SRPT could be $\Omega(\log P)$ worse than optimum.

Leonardi and Raz also considered offline algorithms for the non-preemptive setting in their paper.

Theorem 3 ([8]) *There is a polynomial time off-line algorithm that achieves an approximation ratio of* $O(n^{1/2} \log n/m)$ *for minimizing average flow time on m machines without preemption.*

To prove this result, they give a general technique to convert a preemptive schedule to a non-preemptive one at the loss of an $O(n^{1/2})$ factor in the approximation ratio. They also showed an almost matching lower bound. In particular,

Theorem 4 ([8]) *No polynomial time algorithm for minimizing the total flow time on multiple machines without preemption can have an approximation ratio of* $O(n^{1/3-\varepsilon})$ *for any* $\varepsilon > 0$, *unless P=NP.*

Extensions Since the publication of these results, they have been extended in several directions. Recall that SRPT is both preemptive and migratory. Awerbuch, Azar, Leonardi, and Regev [2] gave an online scheduling algorithm that is non-migratory and still achieves a competitive ratio of $O(\min(\log(n/m), \log P))$. Avrahami and Azar [1] gave an even more restricted $O(\min(\log P, \log(n/m)))$ competitive online algorithm. Their algorithm, in addition to being non-migratory, dispatches a job immediately to a machine upon its arrival. Recently, Garg and Kumar [4,5] have extended these results to a setting where machines have non-uniform speeds. Other related problems and settings such as stretch minimization (defined as the flow time divided by the size of a job), weighted flow time minimization, and the non-clairvoyant setting where the size of a job is not unknown upon its arrival have also been investigated. The reader is referred to a recent survey by Pruhs, Sgall, and Torng [9] for more details.

Applications

The flow time measure considered here is one of the most widely used measures of quality of service, as it corresponds to the amount of time one has to wait to get the job done. The scheduling model considered here arises very naturally when there are multiple resources and several agents that compete for service from these resources. For example, consider a computing system with multiple homogeneous processors where jobs are submitted by users arbitrarily over time. Keeping the average response time low also keeps the frustration levels of the users low. The model is not necessarily limited to computer systems. At a grocery store each cashier can be viewed as a machine, and the users lining up to checkout can be viewed as jobs. The flow time of a user is time spent waiting until she finishes her transaction with the cashier. Of course, in many applications there are additional constraints such as it may be infeasible to preempt jobs, or if customers expect a certain fairness such people might prefer to be serviced in a first come first served manner at a grocery store.

Open Problems

The online algorithm of Leonardi and Raz is also the best-known offline approximation algorithm for the problem. In particular, it is not known whether an $O(1)$ approximation exists even for the case of two machines. Settling this would be very interesting. In related work, Bansal [3] considered the problem of finding non-migratory schedules for a constant number of machines. He gave an algorithm that produces a $(1+\varepsilon)$-approximate solution for any $\varepsilon > 0$ in time $n^{O(\log n/\varepsilon^2)}$. This suggests the possibility of a polynomial time approximation scheme for the problem, at least for the case of a constant number of machines.

Cross References

► Flow Time Minimization
► Multi-level Feedback Queues
► Shortest Elapsed Time First Scheduling

Recommended Reading

1. Avrahami, N., Azar, Y.: Minimizing total flow time and completion time with immediate dispatching. In: Proceedings of 15th SPAA, pp. 11–18. (2003)
2. Awerbuch, B., Azar, Y., Leonardi, S., Regev, O.: Minimizing the flow time without migration. SIAM J. Comput. **31**, 1370–1382 (2002)
3. Bansal, N.: Minimizing flow time on a constant number of machines with preemption. Oper. Res. Lett. **33**, 267–273 (2005)
4. Garg, N., Kumar, A.: Better algorithms for minimizing average flow-time on related machines. In: Proceesings of ICALP, pp. 181–190 (2006)
5. Garg, N., Kumar, A.: Minimizing average flow time on related machines. In: ACM Symposium on Theory of Compuring (STOC), pp. 730–738 (2006)
6. Kellerer, H., Tautenhahn, T., Woeginger, G.J.: Approximability and nonapproximability results for minimizing total flow time on a single machine. SIAM J. Comput. **28**, 1155–1166 (1999)
7. Leonardi, S.: A simpler proof of preemptive flow-time approximation. Approximation and On-line Algorithms. In: Bampis, E. (ed.) Lecture Notes in Computer Science. Springer, Berlin (2003)
8. Leonardi, S., Raz, D.: Approximating total flow time on parallel machines. In: ACM Symposium on Theory of Computing (STOC), pp. 110–119 (1997)
9. Pruhs, K., Sgall, J., Torng, E.: Online scheduling. In: Handbook on Scheduling: Algorithms, Models and Performance Analysis, CRC press (2004). Symposium on Theory of Computing (STOC), pp. 110–119. (1997)

Minimum Geometric Spanning Trees

1999; Krznaric, Levcopoulos, Nilsson

CHRISTOS LEVCOPOULOS
Department of Computer Science, Lund University, Lund, Sweden

Keywords and Synonyms

Minimum length spanning trees; Minimum weight spanning trees; Euclidean minimum spanning trees; MST; EMST

Problem Definition

Let S be a set of n points in d-dimensional real space where $d \geq 1$ is an integer constant. A *minimum spanning tree* (MST) of S is a connected acyclic graph with vertex set S of minimum total edge length. The length of an edge equals the distance between its endpoints under some metric. Under the so-called L_p metric, the distance between two points x and y with coordinates $(x_1, x_2, \dots, x_d)$ and $(y_1, y_2, \dots, y_d)$, respectively, is defined as the pth root of the sum $\sum_{i=1}^{d} |x_i - y_i|^p$.

Key Results

Since there is a very large number of papers concerned with geometric MSTs, only a few of them will be mentioned here.

In the common Euclidean L_2 metric, which simply measures straight-line distances, the MST problem in two dimensions can be solved optimally in time $O(n \log n)$, by using the fact that the MST is a subgraph of the Delaunay triangulation of the input point set. The latter is in turn the dual of the Voronoi diagram of S, for which there exist several $O(n \log n)$-time algorithms. The term "optimally" here refers to the algebraic computation tree model. After computation of the Delaunay triangulation, the MST can be computed in only $O(n)$ additional time, by using a technique by Cheriton and Tarjan [5].

Even for higher dimensions, i. e., when $d > 2$, it holds that the MST is a subgraph of the dual of the Voronoi diagram; however, this fact cannot be exploited in the same way as in the two-dimensional case, because this dual may contain $\Omega(n^2)$ edges. Therefore, in higher dimensions other geometric properties are used to reduce the number of edges which have to be considered. The first subquadratic-time algorithm for higher dimensions was due to Yao [14]. A more efficient algorithm was later proposed by Agarwal et al. [1]. For $d = 3$, their algorithm runs in randomized expected time $O((n \log n)^{4/3})$, and for $d \geq 4$, in expected time $O(n^{2-2/(\lceil d/2 \rceil+1)+\epsilon})$, where ε stands for an arbitrarily small positive constant.

The algorithm by Agarwal et al. builds on exploring the relationship between computing a MST and finding a closest pair between n red points and m blue points, which is called the *bichromatic closest pair* problem. They showed that if $T_d(n, m)$ denotes the time to solve the latter problem, then a MST can be computed in $O(T_d(n, n) \log^d n)$ time. Later, Callahan and Kosaraju [4] improved this bound to $O(T_d(n, n) \log n)$. Both methods achieve running time $O(T_d(n, n))$, if $T_d(n, n) = \Omega(n^{1+\alpha})$, for some $\alpha > 0$. Finally, Krznaric et al. [10] showed that the two problems, i. e., computing a MST and computing the bichromatic closest pair, have the same worst-case time complexity (up to constant factors) in the commonly used algebraic computation tree model, and for any fixed L_p metric. The hardest part to prove is that a MST can be computed in time $O(T_d(n, n))$. The other part, which is that the bichromatic closest pair problem is not harder than computing the MST, is easy to show: if one first computes a MST for the union of the $n + m$ red and blue points, one can then find a closest bichromatic pair in linear time, because at least one such pair has to be connected by some edge of the MST.

The algorithm proposed by Krznaric et al. [10] is based on the standard approach of joining trees in a forest with the shortest edge connecting two different trees, similar to the classical Kruskal's and Prim's MST algorithms for graphs. To reduce the number of candidates to be considered as edges of the MST, the algorithm works in a sequence of phases, where in each phase only edges of equal or similar length are considered, within a factor of 2.

The initial forest is the set S of points, that is, each point of the input constitutes an individual edgeless tree. Then, as long as there is more than one tree in the forest, two trees are merged by producing an edge connecting two nodes, one from each tree. After this procedure, the edges produced comprise a single tree that remains in the forest, and this tree constitutes the output of the algorithm.

Assume that the next edge that the algorithm is going to produce has length l. Each tree T in the forest is partitioned into groups of nodes, each group having a specific node representing the group. The representative node in such a group is called a *leader*. Furthermore, every node in a group including the leader has the property that it lies within distance $\epsilon \cdot l$ from its leader, where ε is a real constant close to zero.

Instead of considering all pairs of nodes which can be candidates for the next edge to produce, first only pairs of leaders are considered. Only if a pair of leaders belong to different trees and the distance between them is approximately l, then the closest pair of points between their two respective groups is computed, using the algorithm for the bichromatic closest pair problem.

Also, the following invariant is maintained: for any phase producing edges of length $\Theta(l)$, and for any leader, there is only a constant number of other leaders at distance $\Theta(l)$. Thus, the total number of pairs of leaders to consider is only linear in the number of leaders.

Nearby leaders for any given leader can be found efficiently by using bucketing techniques and data structures for dynamic closest pair queries [3], together with extra artificial points which can be inserted and removed for probing purposes at various small boxes at distance $\Theta(l)$ from the leader. In order to maintain the invariant, when moving to subsequent phases, one reduces the number of leaders accordingly, as pairs of nearby groups merge into single groups. Another tool which is also needed to consider the right types of pairs is to organize the groups according to the various directions in which there can be new candidate MST edges adjacent to nodes in the group. For details, please see the original paper by Krznaric et al. [10].

There is a special version of the bichromatic closest point problem which was shown by Krznaric et al. [10] to have the same worst-case time complexity as computing a MST: namely, the problem for the special case when both the set of red points and the set of blue points have a very small diameter compared with the distance between the closest bichromatic pair. This ratio can be made arbitrarily small by choosing a suitable ε as the parameter for creating the groups and leaders mentioned above. This fact was exploited in order to derive more efficient algorithms for the three-dimensional case.

For example, in the L_1 metric it is possible to build in time $O(n \log n)$ a special kind of a planar Voronoi diagram for the blue points on a plane separating the blue from the red points having the following property: for each query point q in the half-space including the red points one can use this Voronoi diagram to find in time $O(\log n)$ the blue point which is closest to q under the L_1 metric. (This planar Voronoi diagram can be seen as defined by the vertical projections of the blue points onto the plane containing the diagram, and the size of a Voronoi cell depends on the distance between the corresponding blue point and the plane.) So, by using subsequently every red point as a query point for this data structure, one can solve the bichromatic closest pair problem for such well-separated red–blue sets in total $O(n \log n)$ time.

By exploiting and building upon this idea, Krznaric et al. [10] showed how to find a MST of S in optimal $O(n \log n)$ time under the L_1 and L_∞ metrics when $d = 3$. This is an improvement over previous bounds due to Gabow et al. [9] and Bespamyatnikh [2], who proved that, for $d = 3$, a MST can be computed in $O(n \log n \log \log n)$ time under the L_1 and L_∞ metrics.

The main results of Krznaric et al. [10] are summarized in the following theorem.

Theorem *In the algebraic computation tree model, for any fixed L_p metric, and for any fixed number of dimensions, computing the MST has the same worst-case complexity, within constant factors, as solving the bichromatic closest pair problem. Moreover, for three-dimensional space under the L_1 and L_∞ metrics, the MST (as well as the bichromatic closest pair) can be computed in optimal $O(n \log n)$ time.*

Approximate and Dynamic Solutions

Callahan and Kosaraju [4] showed that a spanning tree of length within a factor $1 + \epsilon$ from that of a MST can be computed in time $O(n(\log n + \epsilon^{-d/2} \log \epsilon^{-1}))$. Approximation algorithms with worse tradeoff between time and quality had earlier been developed by Clarkson [6], Vaidya [13] and Salowe [12]. In addition, if the input point set is supported by certain basic data structures, then the approximate length of a MST can be computed in randomized sublinear time [7]. Eppstein [8] gave fully dynamic algorithms that maintain a MST when points are inserted or deleted.

Applications

MSTs belong to the most basic structures in computational geometry and in graph theory, with a vast number of applications.

Open Problems

Although the complexity of computing MSTs is settled in relation to computing bichromatic closest pairs, this means also that it remains open for all cases where the complexity of computing bichromatic closest pairs remains open, e. g., when the number of dimensions is greater than 3.

Experimental Results

Narasimhan and Zachariasen [11] have reported experiments with computing geometric MSTs via well-separated pair decompositions.

Cross References

▶ Degree-Bounded Trees
▶ Fully Dynamic Minimum Spanning Trees
▶ Greedy Set-Cover Algorithms
▶ Max Leaf Spanning Tree
▶ Minimum Spanning Trees
▶ Parallel Connectivity and Minimum Spanning Trees
▶ Randomized Minimum Spanning Tree
▶ Rectilinear Spanning Tree
▶ Rectilinear Steiner Tree
▶ Steiner Forest
▶ Steiner Trees

Recommended Reading

1. Agarwal, P.K., Edelsbrunner, H., Schwarzkopf, O., Welzl, E.: Euclidean minimum spanning trees and bichromatic closest pairs. Discret. Comput. Geom. **6**, 407–422 (1991)

2. Bespamyatnikh, S.: On Constructing Minimum Spanning Trees in R_1^k. Algorithmica **18**(4), 524–529 (1997)
3. Bespamyatnikh, S.: An Optimal Algorithm for Closest-Pair Maintenance. Discret. Comput. Geom. **19**(2), 175–195 (1998)
4. Callahan, P.B., Kosaraju, S.R.: Faster Algorithms for Some Geometric Graph Problems in Higher Dimensions. In: SODA 1993, pp. 291–300
5. Cheriton, D.and Tarjan, R.E.: Finding Minimum Spanning Trees. SIAM J. Comput. **5**(4), 724–742 (1976)
6. Clarkson, K.L.: Fast Expected-Time and Approximation Algorithms for Geometric Minimum Spanning Trees. In: Proc. STOC 1984, pp. 342–348
7. Czumaj, A., Ergün, F., Fortnow, L., Magen, A., Newman, I., Rubinfeld, R., Sohler, C.: Approximating the Weight of the Euclidean Minimum Spanning Tree in Sublinear Time. SIAM J. Comput. **35**(1), 91–109 (2005)
8. Eppstein, D.: Dynamic Euclidean Minimum Spanning Trees and Extrema of Binary Functions. Discret. Comput. Geom. **13**, 111–122 (1995)
9. Gabow, H.N., Bentley, J.L., Tarjan, R.E.: Scaling and Related Techniques for Geometry Problems. In: STOC 1984, pp. 135–143
10. Krznaric, D., Levcopoulos, C., Nilsson, B.J.: Minimum Spanning Trees in d Dimensions. Nord. J. Comput. **6**(4), 446–461 (1999)
11. Narasimhan, G., Zachariasen, M.: Geometric Minimum Spanning Trees via Well-Separated Pair Decompositions. ACM J. Exp. Algorithms **6**, 6 (2001)
12. Salowe, J.S.: Constructing multidimensional spanner graphs. Int. J. Comput. Geom. Appl. **1**(2), 99–107 (1991)
13. Vaidya, P.M.: Minimum Spanning Trees in k-Dimensional Space. SIAM J. Comput. **17**(3), 572–582 (1988)
14. Yao, A.C.: On Constructing Minimum Spanning Trees in k-Dimensional Spaces and Related Problems. SIAM J. Comput. **11**(4), 721–736 (1982)

Minimum *k*-Connected Geometric Networks

2000; Czumaj, Lingas

ARTUR CZUMAJ[1], ANDRZEJ LINGAS[2]

[1] DIMAP and Computer Science, University of Warwick, Coventry, UK

[2] Department of Computer Science, Lund University, Lund, Sweden

Keywords and Synonyms

Geometric graphs; Euclidean graphs

Problem Definition

The following classical optimization problem is considered: for a given undirected weighted geometric network, find its minimum-cost sub-network that satisfies a priori given multi-connectivity requirements.

Notations

Let $G = (V, E)$ be a *geometric network*, whose vertex set V corresponds to a set of n points in $\mathbb{R}^d$ for certain integer d, $d \geq 2$, and whose edge set E corresponds to a set of straight-line segments connecting pairs of points in V. G is called complete if E connects all pairs of points in V.

The cost $\delta(x, y)$ of an edge connecting a pair of points $x, y \in \mathbb{R}^d$ is equal to the Euclidean distance between points x and y, that is, $\delta(x, y) = \sqrt{\sum_{i=1}^{d}(x_i - y_i)^2}$, where $x = (x_1, \ldots, x_d)$ and $y = (y_1, \ldots, y_d)$. More generally, the cost $\delta(x, y)$ could be defined using other norms, such as ℓ_p norms for any $p > 1$, i. e., $\delta(x, y) = \left(\sum_{i=1}^{p}(x_i - y_i)^p\right)^{1/p}$. The cost of the network is equal to the sum of the costs of the edges of the network, $\text{cost}(G) = \sum_{(x,y)\in E} \delta(x, y)$.

A network $G = (V, E)$ is *spanning* a set S of points if $V = S$. $G = (V, E)$ is *k-vertex-connected* if for any set $U \subseteq V$ of fewer than k vertices, the network $(V \setminus U,\ E \cap ((V \setminus U) \times (V \setminus U)))$ is connected. Similarly, G is *k-edge-connected* if for any set $\mathcal{E} \subseteq E$ of fewer than k edges, the network $(V, E \setminus \mathcal{E})$ is connected.

The (Euclidean) Minimum-Cost k-Vertex Connected Spanning Network Problem For a given set S of n points in the Euclidean space $\mathbb{R}^d$, find a minimum-cost k-vertex connected Euclidean network spanning points in S.

The (Euclidean) Minimum-Cost k-Edge Connected Spanning Network Problem For a given set S of n points in the Euclidean space $\mathbb{R}^d$, find a minimum-cost k-edge connected Euclidean network spanning points in S.

A variant that allows *parallel edges* is also considered:

The (Euclidean) Minimum-Cost k-Edge Connected Spanning Multi-Network Problem For a given set S of n points in the Euclidean space $\mathbb{R}^d$, find a minimum-cost k-edge connected Euclidean multi-network spanning points in S (where the multi-network can have parallel edges).

The concept of minimum-cost k-connectivity naturally extends to include that of *Euclidean Steiner k-connectivity* by allowing the use of additional vertices, called *Steiner points*. For a given set S of points in $\mathbb{R}^d$, a geometric network G is a *Steiner k-vertex connected (or, Steiner k-edge connected) for S* if the vertex set of G is a *superset* of S and for every pair of points from S there are k internally vertex-disjoint (edge-disjoint, respectively) paths connecting them in G.

The (Euclidean) Minimum-Cost Steiner k-Vertex/Edge Connectivity Problem Find a minimum-cost network on a superset of S that is Steiner k-vertex/edge connected for S.

Note that for $k = 1$, it is simply the *Steiner minimal tree* problem, which has been very extensively studied in the literature (see, e. g., [14]).

In a more general formulation of multi-connectivity graph problems, non-uniform connectivity constraints have to be satisfied.

The Survivable Network Design Problem For a given set S of points in $\mathbb{R}^d$ and a connectivity requirement function $r : S \times S \to \mathbb{N}$, find a minimum-cost geometric network spanning points in S such that for any pair of vertices $p, q \in S$ the sub-network has $r_{p,q}$ internally vertex-disjoint (or edge-disjoint, respectively) paths between p and q.

In many applications of this problem, often regarded as the most interesting ones [9,13], the connectivity requirement function is specified with the help of a one-argument function which assigns to each vertex p its connectivity type $r_v \in \mathbb{N}$. Then, for any pair of vertices $p, q \in S$, the connectivity requirement $r_{p,q}$ is simply given as $\min\{r_p, r_q\}$ [12,13,17,18]. This includes the *Steiner tree problem* (see, e. g., [2]), in which $r_p \in \{0, 1\}$ for any vertex $p \in S$.

A *polynomial-time approximation scheme (PTAS)* is a family of algorithms $\{\mathcal{A}_\varepsilon\}$ such that, for each fixed $\varepsilon > 0$, $\mathcal{A}_\varepsilon$ runs in time polynomial in the size of the input and produces a $(1 + \varepsilon)$-approximation.

Related Work

For a very extensive presentation of results concerning problems of finding minimum-cost k-vertex- and k-edge-connected spanning subgraphs, non-uniform connectivity, connectivity augmentation problems, and geometric problems, see [1,3,11,15].

Despite the practical relevance of the multi-connectivity problems for geometrical networks and the vast amount of practical heuristic results reported (see, e. g., [12,13,17,18]), very little theoretical research had been done towards developing efficient approximation algorithms for these problems until a few years ago. This contrasts with the very rich and successful theoretical investigations of the corresponding problems in general metric spaces and for general weighted graphs. And so, until 1998, even for the simplest and most fundamental multi-connectivity problem, that of finding a minimum-cost 2-vertex connected network spanning a given set of points in the Euclidean plane, obtaining approximations achieving better than a $\frac{3}{2}$ ratio had been elusive (the ratio $\frac{3}{2}$ is the best polynomial-time approximation ratio known for general networks whose weights satisfy the triangle inequality [8]; for other results, see e. g., [4,15]).

Key Results

The first result is an extension of the well-known $\mathcal{NP}$-hardness result of minimum-cost 2-connectivity in general graphs (see, e. g., [10]) to geometric networks.

Theorem 1 *The problem of finding a minimum-cost 2-vertex/edge connected geometric network spanning a set of n points in the plane is $\mathcal{NP}$-hard.*

Next result shows that if one considers the minimum-cost multi-connectivity problems in an enough high dimension, the problems become APX-hard.

Theorem 2 ([6]) *There exists a constant $\xi > 0$ such that it is $\mathcal{NP}$-hard to approximate within $1 + \xi$ the minimum-cost 2-connected geometric network spanning a set of n points in $\mathbb{R}^{\lceil \log_2 n \rceil}$.*

This result extends also to any ℓ_p norm.

Theorem 3 ([6]) *For and integer $d \geq \log n$ and for any fixed $p \geq 1$ there exists a constant $\xi > 0$ such that it is $\mathcal{NP}$-hard to approximate within $1 + \xi$ the minimum-cost 2-connected network spanning a set of n points in the ℓ_p metric in $\mathbb{R}^d$.*

Since the minimum-cost multi-connectivity problems are hard, the research turned into the study of approximation algorithms. By combining some of the ideas developed for the polynomial-time approximation algorithms for TSP due to Arora [2] (see also [16]) together with several new ideas developed specifically for the multi-connectivity problems in geometric networks, Czumaj and Lingas obtained the following results.

Theorem 4 ([5,6]) *Let k and d be any integers, $k, d \geq 2$, and let ε be any positive real. Let S be a set of n points in $\mathbb{R}^d$. There is a randomized algorithm that in time $n \cdot (\log n)^{(kd/\varepsilon)^{\mathcal{O}(d)}} \cdot 2^{2^{(kd/\varepsilon)^{\mathcal{O}(d)}}}$ with probability at least 0.99 finds a k-vertex-connected (or k-edge-connected) spanning network for S whose cost is at most $(1 + \varepsilon)$-time optimal.*

Furthermore, this algorithm can be derandomized in polynomial-time to return a k-vertex-connected (or k-edge-connected) spanning network for S whose cost is at most $(1 + \varepsilon)$ times the optimum.

Observe that when all d, k, and ε are constant, then the running-times are $n \cdot \log^{\mathcal{O}(1)} n$.

The results in Theorem 4 give a PTAS for small values of k and d.

Theorem 5 (PTAS for vertex/edge-connectivity [6,5]) *Let $d \geq 2$ be any constant integer. There is a certain positive constant $c < 1$ such that for all k such that $k \leq (\log\log n)^c$, the problems of finding a minimum-cost k-vertex-connected spanning network and a k-edge-connected spanning network for a set of points in $\mathbb{R}^d$ admit PTAS.*

The next theorem deals with multi-networks where feasible solutions are allowed to use parallel edges.

Theorem 6 ([5]) *Let k and d be any integers, $k, d \geq 2$, and let ε be any positive real. Let S be a set of n points in $\mathbb{R}^d$. There is a randomized algorithm that in time $n \cdot \log n \cdot (d/\varepsilon)^{\mathcal{O}(d)} + n \cdot 2^{2^{(k^{\mathcal{O}(1)} \cdot (d/\varepsilon)^{\mathcal{O}(d^2)})}}$, with probability at least 0.99 finds a k-edge-connected spanning multi-network for S whose cost is at most $(1+\varepsilon)$ times the optimum. The algorithm can be derandomized in polynomial-time.*

Combining this theorem with the fact that parallel edges can be eliminated in case $k = 2$, one obtains the following result for 2-connectivity in networks.

Theorem 7 (Approximation schemes for 2-connected graphs, [5]) *Let d be any integer, $d \geq 2$, and let ε be any positive real. Let S be a set of n points in $\mathbb{R}^d$. There is a randomized algorithm that in time $n \cdot \log n \cdot (d/\varepsilon)^{\mathcal{O}(d)} + n \cdot 2^{(d/\varepsilon)^{\mathcal{O}(d^2)}}$, with probability at least 0.99 finds a 2-vertex-connected (or 2-edge-connected) spanning network for S whose cost is at most $(1+\varepsilon)$ times the optimum. This algorithm can be derandomized in polynomial-time.*

For constant d the running time of the randomized algorithms is $n \log n \cdot (1/\varepsilon)^{\mathcal{O}(1)} + 2^{(1/\varepsilon)^{\mathcal{O}(1)}}$.

Theorem 8 ([7]) *Let d be any integer, $d \geq 2$, and let ε be any positive real. Let S be a set of n points in $\mathbb{R}^d$. There is a randomized algorithm that in time $n \cdot \log n \cdot (d/\varepsilon)^{\mathcal{O}(d)} + n \cdot 2^{(d/\varepsilon)^{\mathcal{O}(d^2)}} + n \cdot 2^{2^{d^{d^{\mathcal{O}(1)}}}}$, with probability at least 0.99 finds a Steiner 2-vertex-connected (or 2-edge-connected) spanning network for S whose cost is at most $(1+\varepsilon)$ times the optimum. This algorithm can be derandomized in polynomial-time.*

Theorem 9 ([7]) *Let d be any integer, $d \geq 2$, and let ε be any positive real. Let S be a set of n points in $\mathbb{R}^d$. There is a randomized algorithm that in time $n \cdot \log n \cdot (d/\varepsilon)^{\mathcal{O}(d)} + n \cdot 2^{(d/\varepsilon)^{\mathcal{O}(d^2)}} + n \cdot 2^{2^{d^{d^{\mathcal{O}(1)}}}}$, with probability at least 0.99 gives a $(1+\varepsilon)$-approximation for the geometric network survivability problem with $r_v \in \{0,1,2\}$ for any $v \in V$. This algorithm can be derandomized in polynomial-time.*

Applications

Multi-connectivity problems are central in algorithmic graph theory and have numerous applications in computer science and operation research, see, e. g., [1,13, 11,18]. They also play very important role in the design of networks that arise in practical situations, see, e. g., [1,13]. Typical application areas include telecommunication, computer and road networks. Low degree connectivity problems for geometrical networks in the plane can often closely *approximate* such practical connectivity problems (see, e. g., the discussion in [13,17,18]). The survivable network design problem in geometric networks also arises in many applications, e. g., in telecommunication, communication network design, VLSI design, etc. [12,13,17,18].

Open Problems

The results discussed above lead to efficient algorithms only for small connectivity requirements k; the running-time is polynomial only for the value of k up to $(\log\log n)^c$ for certain positive constant $c < 1$. It is an interesting open problem if one can obtain polynomial-time approximation schemes algorithms also for large values of k.

It is also an interesting open problem if the multi-connectivity problems in geometric networks can have practically fast approximation schemes.

Cross References

► Euclidean Traveling Salesperson Problem
► Minimum Geometric Spanning Trees

Recommended Reading

1. Ahuja, R.K., Magnanti, T.L., Orlin, J.B., Reddy, M.R.: Applications of network optimization. In: Handbooks in Operations Research and Management Science, vol. 7, Network Models, chapter 1, pp. 1–83. North-Holland, Amsterdam (1995)
2. Arora, S.: Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. J. ACM **45**(5), 753–782 (1998)
3. Berger, A., Czumaj, A., Grigni, M., Zhao, H.: Approximation schemes for minimum 2-connected spanning subgraphs in weighted planar graphs. Proc. 13th Annual European Symposium on Algorithms, pp. 472–483. (2005)
4. Cheriyan, J., Vetta, A.: Approximation algorithms for network design with metric costs. Proc. 37th Annual ACM Symposium on Theory of Computing, Baltimore, 22–24 May 2005, pp. 167–175. (2005)
5. Czumaj, A., Lingas, A.: Fast approximation schemes for Euclidean multi-connectivity problems. Proc. 27th Annual International Colloquium on Automata, Languages and Programming, Geneva, 9–15 July 2000, pp. 856–868

6. Czumaj, A., Lingas, A.: On approximability of the minimum-cost *k*-connected spanning subgraph problem. Proc. 10th Annual ACM-SIAM Symposium on Discrete Algorithms, Baltimore, 17–19 January 1999, pp. 281–290
7. Czumaj, A., Lingas, A., Zhao, H.: Polynomial-time approximation schemes for the Euclidean survivable network design problem. Proc. 29th Annual International Colloquium on Automata, Languages and Programming, Malaga, 8–13 July 2002, pp. 973–984
8. Frederickson, G.N., JáJá, J.: On the relationship between the biconnectivity augmentation and Traveling Salesman Problem. Theor. Comput. Sci. **19**(2), 189–201 (1982)
9. Gabow, H.N., Goemans, M.X., Williamson, D.P.: An efficient approximation algorithm for the survivable network design problem. Math. Program. Ser. B **82**(1–2), 13–40 (1998)
10. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-completeness. Freeman, New York, NY (1979)
11. Goemans, M.X., Williamson, D.P.: The primal-dual method for approximation algorithms and its application to network design problems. In: Hochbaum, D. (ed.) Approximation Algorithms for $\mathcal{NP}$-Hard Problems, Chapter 4, pp. 144–191. PWS Publishing Company, Boston (1996)
12. Grötschel, M., Monma, C.L., Stoer, M.: Computational results with a cutting plane algorithm for designing communication networks with low-connectivity constraints. Oper. Res. **40**(2), 309–330 (1992)
13. Grötschel, M., Monma, C.L., Stoer, M.: Design of survivable networks. In: Handbooks in Operations Research and Management Science, vol. 7, Network Models, chapter 10, pp. 617–672. North-Holland, Amsterdam (1995)
14. Hwang, F.K., Richards, D.S., Winter, P.: The Steiner Tree Problem. North-Holland, Amsterdam (1992)
15. Khuller, S.: Approximation algorithms for finding highly connected subgraphs. In: Hochbaum, D. (ed.) Approximation Algorithms for $\mathcal{NP}$-Hard Problems, Chapter 6, pp. 236–265. PWS Publishing Company, Boston (1996)
16. Mitchell, J.S.B.: Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric TSP, *k*-MST, and related problems. SIAM J. Comput. **28**(4), 1298–1309 (1999)
17. Monma, C.L., Shallcross, D.F.: Methods for designing communications networks with certain two-connected survivability constraints. Operat. Res. **37**(4), 531–541 (1989)
18. Stoer, M.: Design of Survivable Networks. Springer, Berlin (1992)

Minimum Makespan on Unrelated Machines

1990; Lenstra, Shmoys, Tardos

MAXIM SVIRIDENKO
IBM Research, IBM, Yorktown Heights, NY, USA

Keywords and Synonyms

Schedule of minimum length on different machines

Problem Definition

Consider the following scheduling problem. There are m parallel machines and n independent jobs. Each job is to be assigned to one of the machines. The processing of job j on machine i requires p_{ij} units of time. The objective is to find a schedule that minimizes the makespan, defined to be the time by which all jobs are completed. This problem is denoted $R||C_{\max}$ using standard scheduling notation terminology [6].

There are few important special cases of the problem: the restricted assignment problem with $p_{ij} \in \{1, \infty\}$, the identical parallel machines with $p_{ij} = p_j$ and the uniform parallel machines $p_{ij} = p_j/s_i$ where $s_i > 0$ is a speed of machine i. These problems are denoted $R|p_{ij} \in \{1, \infty\}|C_{\max}$, $P||C_{\max}$ and $Q||C_{\max}$, respectively. Two later problems admit polynomial time approximation schemes [4,5].

Consider the following integer programming formulation of the feasibility problem that finds a feasible assignment of jobs to machines with makespan at most T

$$\sum_{i=1}^{m} x_{ij} = 1\,, \quad j = 1, \dots, n\,, \tag{1}$$

$$\sum_{j=1}^{n} p_{ij} x_{ij} \le T\,, \quad i = 1, \dots, m\,, \tag{2}$$

$$x_{ij} = 0\,, \quad \text{if} \quad p_{ij} > T\,, \tag{3}$$

$$x_{ij} \in \{0, 1\}\,, \quad \forall i, j\,. \tag{4}$$

The variable $x_{ij} = 1$ if job j is assigned to machine i and $x_{ij} = 0$, otherwise. The constraint (1) corresponds to job assignments. The constraint (2) bounds the total processing time of jobs assigned to one machine. The constraint (3) forbids an assignment of a job to a machine if its processing time is larger than the target makespan T.

Key Results

Theorem 1 (Rounding Theorem) *Consider the linear programming relaxation of the integer program (1)–(4) by relaxing the integrality constraint (4) with the constraint*

$$x_{ij} \ge 0\,, \quad \forall i, j\,. \tag{5}$$

If the linear program (1)–(3),(5) has a feasible solution for some value of parameter $T = T^*$ then there exists a feasible solution to an integer program (1)–(4) with parameter $T = T^* + p_{\max}$ where $p_{\max} = \max_{i,j} p_{ij}$ and such a solution can be found in polynomial time.

The idea of the proof is to start with a basic feasible solution of the linear program (1)–(3),(5). The properties of basic solutions imply the bound on the number of fractional variables which in turn implies that the bipartite graph defined between jobs and machines with edges corresponding to fractional variables has a very special structure. Lenstra, Shmoys and Tardos [7] show that it is possible to round fractional edges in such a way that each machine node has at most one edge (variable) rounded up which implies the bound on the makespan.

The Theorem 1 combined with binary search on parameter T implies

Corollary 1 *There is a 2-approximation algorithm for the makespan minimization problem on unrelated parallel machines that runs in time polynomial in the input size.*

Lenstra, Shmoys and Tardos [7] proved an inapproximability result that is valid even for the case of the restricted assignment problem

Theorem 2 *For every $\rho < 3/2$ there does not exist a polynomial ρ-approximation algorithm for the makespan minimization problem on unrelated parallel machines unless $P = NP$.*

Generalizations

A natural generalization of the scheduling problem is to add additional resource requirements $\sum_{i,j} c_{ij} x_{ij} \le B$, i. e. there is a cost c_{ij} associated with assigning job i to machine j. The goal is to find an assignment of jobs to machines of total cost at most B minimizing the makespan. This problem is known under the name of the generalized assignment problem. Shmoys and Tardos [8] proved an analogous Rounding Theorem leading to a 2-approximation algorithm.

Even more general problem arises when each machine i has few modes $s = 1, \dots, k$ to process job j. Each mode has the processing time p_{ijs} and the cost c_{ijs} associated with it. The goal is to find a an assignment of jobs to machines and modes of total cost at most B minimizing the makespan. Consider the following analogous integer programming formulation of the problem

$$\sum_{i=1}^{m} \sum_{s=0}^{k} x_{ijs} = 1\,, \quad j = 1, \dots, n\,, \tag{6}$$

$$\sum_{j=1}^{n} \sum_{s=0}^{k} x_{ijs}\, p_{ijs} \le T\,, \quad i = 1, \dots, m\,, \tag{7}$$

$$\sum_{j=1}^{n} \sum_{i=1}^{m} \sum_{s=0}^{k} x_{ijs}\, c_{ijs} \le B\,, \tag{8}$$

$$x_{ijs} = 0\,, \quad \text{if } p_{ijs} > T\,, \tag{9}$$

$$x_{ijs} \in \{0, 1\}\,, \quad \forall\, i, j, s\,. \tag{10}$$

Theorem 3 (General Rounding Theorem) *Consider the linear programming relaxation of the integer program (6)–(10) by relaxing the integrality constraint (10) with the constraint*

$$x_{ijs} \ge 0\,, \quad \forall i, j, s\,. \tag{11}$$

If linear program (6)–(9),(11) has a feasible solution for some value of parameter $T = T^*$ then there exists a feasible solution to the integer program (6)–(10) with parameter $T = T^* + p_{\max}$ where $p_{\max} = \max_{i,j,s} p_{ijs}$ and such a solution can be found in polynomial time.

The randomized version of this Theorem was originally proved by Gandhi, Khuller, Parthasarathy and Srinivasan [2]. The deterministic version appeared first in [3].

Applications

Unrelated parallel machine scheduling is one of the basic scheduling models with a lot of industrial applications, see for example [1, 9]. The rounding Theorem by Lenstra, Shmoys, Tardos and its generalizations have found numerous applications applications in design and analysis of approximation algorithms where quite often generalized assignment problem needs to be solved as a subroutine.

Open Problems

The most exciting open problem is to close the gap between positive (Corollary 1) and negative (Theorem 2) results for $R||C_{\max}$. A very simple example shows that the integrality gap of the linear programming relaxation (1)–(3),(5) is 2 and therefore there is a need for a stronger LP to improve upon 2-approximation.

This example consists of $m(T-1)$ jobs such that for each $i \in 1, \dots, m$, processing time $p_{ij} = 1$ for $j = (T-1)(i-1)+1, \dots, (T-1)i$ and $p_{ij} = \infty$ otherwise. In other words, each machine has $T-1$ jobs with unit processing time, that cannot be processed on any other machine. Additionally, there is one large job b with processing time $p_{ib} = T$ for $i = 1, \dots, m$.

One way to define a stronger LP is to define variables x_{iS} for each possible set S of jobs. The variable $x_{iS} = 1$ if the set S of jobs is assigned to be processed on machine i. The set of jobs S is feasible for machine i if $\sum_{j \in S} p_{ij} \le T$. Let C_i be the set of feasible sets for machine i. Consider the following linear programming relaxation:

$$\sum_{i, S \in C_i : j \in S} x_{iS} = 1\,, \quad j = 1, \dots, n\,, \tag{12}$$

$$\sum_{S \in C_i} x_{iS} = 1\,, \quad i = 1, \ldots, m\,, \tag{13}$$

$$x_{iS} \geq 0\,, \quad \forall\, i, S \in C_i\,. \tag{14}$$

The integrality gap of this linear program is also 2 for general unrelated parallel machine scheduling but it is open for the special case of restricted assignment problem.

Cross References

▶ Flow Time Minimization
▶ List Scheduling
▶ Load Balancing
▶ Minimum Flow Time
▶ Minimum Weighted Completion Time

Recommended Reading

1. Jeng-Fung, C.: Unrelated parallel machine scheduling with secondary resource constraints. Int. J. Adv. Manuf. Technol. **26**, 285–292 (2005)
2. Gandhi, R., Khuller, S., Parthasarathy, S., Srinivasan, A.: Dependent rounding and its applications to approximation algorithms. J. ACM **53**(3), 324–360 (2006)
3. Grigoriev, A., Sviridenko, M., Uetz, M.: Machine scheduling with resource dependent processing times. Math. Program. **110**(1B), 209–228 (2002)
4. Hochbaum, D.S., Shmoys, D.B.: Using dual approximation algorithms for scheduling problems: theoretical and practical results. J. Assoc. Comput. Mach. **34**(1), 144–162 (1987)
5. Hochbaum, D.S., Shmoys, D.B.: A polynomial approximation scheme for scheduling on uniform processors: using the dual approximation approach. SIAM J. Comput. **17**(3), 539–551 (1988)
6. Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., Shmoys, D.B.: Sequencing and Scheduling: Algorithms and Complexity. In: Graves, S.C., Rinnooy Kan, A.H.G., Zipkin, P.H. (eds.) Logistics of Production and Inventory. Handbooks in Operations Research and Management Science, vol. 4, pp. 445–522. North–Holland, Amsterdam (1993)
7. Lenstra, J.K., Shmoys, D., Tardos, E.: Approximation algorithms for scheduling unrelated parallel machines. Math. Program. **46**(3A), 259–271 (1990)
8. Shmoys, D., Tardos, E.: An approximation algorithm for the generalized assignment problem. Math. Program. **62**(3A), 461–474 (1993)
9. Yu, L., Shih, H., Pfund, M., Carlyle, W., Fowler, J.: Scheduling of unrelated parallel machines: an application to PWB manufacturing. IIE Trans. **34**, 921–931 (2002)

Minimum Spanning Trees

2002; Pettie, Ramachandran

Seth Pettie
Department of Computer Science, University of Michigan, Ann Arbor, Ann Arbor, MI, USA

Keywords and Synonyms

Minimal spanning tree; Minimum weight spanning tree; Shortest spanning tree

Problem Definition

The *minimum spanning tree* (MST) problem is, given a connected, weighted, and undirected graph $G = (V, E, w)$, to find the tree with minimum total *weight* spanning all the vertices V. Here $w\colon E \to \mathbb{R}$ is the weight function. The problem is frequently defined in geometric terms, where V is a set of points in d-dimensional space and w corresponds to Euclidean distance. The main distinction between these two settings is the form of the input. In the graph setting the input has size $O(m + n)$ and consists of an enumeration of the $n = |V|$ vertices and $m = |E|$ edges and edge weights. In the geometric setting the input consists of an enumeration of the coordinates of each point ($O(dn)$ space): all $\binom{V}{2}$ edges are implicitly present and their weights implicit in the point coordinates. See [16] for a discussion of the Euclidean minimum spanning tree problem.

History

The MST problem is generally recognized [7,12] as one of the first combinatorial problems studied specifically from an algorithmic perspective. It was formally defined by Borůvka in 1926 [1] (predating the fields of computability theory and combinatorial optimization, and even much of graph theory) and since his initial algorithm there has been a sustained interest in the problem. The MST problem has motivated research in matroid optimization [3] and the development of efficient data structures, particularly priority queues (aka heaps) and disjoint set structures [2,18].

Related Problems

The MST problem is frequently contrasted with the *traveling salesman* and *minimum Steiner tree* problems [6]. A Steiner tree is a tree that may span any *superset* of the given points; that is, additional points may be introduced that reduce the weight of the minimum spanning tree. The traveling salesman problem asks for a tour (cycle) of the vertices with minimum total length. The generalization of the MST problem to directed graphs is sometimes called the *minimum branching* [5]. Whereas the undirected and directed versions of the MST problem are solvable in polynomial time, traveling salesman and minimum Steiner tree are NP-complete [6].

Optimality Conditions

A *cut* is a partition (V', V'') of the vertices V. An edge (u, v) *crosses* the cut (V', V'') if $u \in V'$ and $v \in V''$. A sequence $(v_0, v_1, \ldots, v_{k-1}, v_0)$ is a *cycle* if $(v_i, v_{i+1(\text{mod } k)}) \in E$ for $0 \leq i < k$.
The correctness of all MST algorithms is established by appealing to the dual *cut* and *cycle* properties, also known as the blue rule and red rule [18].

Cut Property An edge is in some minimum spanning tree if and only if it is the lightest edge crossing some cut.

Cycle Property An edge is not in any minimum spanning tree if and only if it is the sole heaviest edge on some cycle.

It follows from the cut and cycle properties that if the edge weights are unique then there is a unique minimum spanning tree, denoted $\mathbf{MST}(G)$. Uniqueness can always be enforced by breaking ties in any consistent manner. MST algorithms frequently appeal to a useful corollary of the cut and cycle properties called the *contractibility* property. Let $G \setminus C$ denote the graph derived from G by contracting the subgraph C, that is, C is replaced by a single vertex c and all edges incident to exactly one vertex in C become incident to c; in general $G \setminus C$ may have more than one edge between two vertices.

Contractibility Property If C is a subgraph such that for all pairs of edges e and f with exactly one endpoint in C, there exists a path $P \subseteq C$ connecting ef with each edge in P lighter than either e or f, then C is *contractible*. For any contractible C it holds that $\mathbf{MST}(G) = \mathbf{MST}(C) \cup \mathbf{MST}(G \setminus C)$.

The Generic Greedy Algorithm

Until recently all MST algorithms could be viewed as mere variations on the following generic greedy MST algorithm. Let $\mathcal{T}$ consist initially of n trivial trees, each containing a single vertex of G. Repeat the following step $n - 1$ times. Choose any $T \in \mathcal{T}$ and find the minimum weight edge (u, v) with $u \in T$ and v in a different tree, say $T' \in \mathcal{T}$. Replace T and T' in $\mathcal{T}$ with the single tree $T \cup \{(u, v)\} \cup T'$. After $n - 1$ iterations $\mathcal{T} = \{\mathbf{MST}(G)\}$. By the cut property every edge selected by this algorithm is in the MST.

Modeling MST Algorithms

Another corollary of the cut and cycle properties is that the set of minimum spanning trees of a graph is determined solely by the relative order of the edge weights—their specific numerical values are not relevant. Thus, it is natural to model MST algorithms as *binary decision trees*, where nodes of the decision tree are identified with edge weight comparisons and the children of a node correspond to the possible outcomes of the comparison. In this decision tree model a trivial lower bound on the *time* of the optimal MST algorithm is the *depth* of the optimal decision tree.

Key Results

The primary result of [14] is an explicit MST algorithm that is *provably* optimal even though its asymptotic running time is currently unknown.

Theorem 1 *There is an explicit, deterministic minimum spanning tree algorithm whose running time is on the order of $D_{MST}(m, n)$, where m is the number of edges, n the number of vertices, and $D_{MST}(m, n)$ the maximum depth of an optimal decision tree for any m-edge n-node graph.*

It follows that the Pettie–Ramachandran algorithm [14] is asymptotically no worse than *any* MST algorithm that deduces the solution through edge weight comparisons. The best known upper bound on $D_{MST}(m, n)$ is $O(m\alpha(m, n))$, due to Chazelle [2]. It is trivially $\Omega(m)$.
Let us briefly describe how the Pettie–Ramachandran algorithm works. An (m, n) instance is a graph with m edges and n vertices. Theorem 1 is proved by giving a linear time *decomposition* procedure that reduces any (m, n) instance of the MST problem to instances of size $(m^*, n^*), (m_1, n_1), \ldots, (m_s, n_s)$, where $m = m^* + \sum_i m_i$, $n = \sum_i n_i$, $n^* \leq n/\log\log\log n$ and each $n_i \leq \log\log\log n$. The (m^*, n^*) instance can be solved in $O(m + n)$ time with existing MST algorithms [2]. To solve the other instances the Pettie–Ramachandran algorithm performs a brute-force search to find the minimum depth decision tree for *every* graph with at most $\log\log\log n$ vertices. Once these decision trees are found the remaining instances are solved in $O(\sum_i D_{MST}(m_i, n_i)) = O(D_{MST}(m, n))$ time. Due to the restricted size of these instances ($n_i \leq \log\log\log n$) the time for a brute force search is a negligible $o(n)$. The decomposition procedure makes use of Chazelle's *soft heap* [2] (an approximate priority queue) and an extension of the contractibility property.

Approximate Contractibility Let G' be derived from G by *increasing* the weight of some edges. If C is contractible w.r.t. G' then $\mathbf{MST}(G) = \mathbf{MST}(\mathbf{MST}(C) \cup \mathbf{MST}(G \setminus C) \cup E^*)$, where E^* is the set of edges with increased weights.

A secondary result of [14] is that the running time of the optimal algorithm is actually linear on nearly every graph topology, under any permutation of the edge weights.

Theorem 2 *Let G be selected uniformly at random from the set of all n-vertex, m-edge graphs. Then regardless of the edge weights,* **MST**(G) *can be found in* $O(m + n)$ *time with probability* $1 - 2^{-\Omega(m/\alpha^2)}$, *where* $\alpha = \alpha(m, n)$ *is the slowly growing inverse-Ackermann function.*

Theorem 1 should be contrasted with the results of Karger, Klein, and Tarjan [9] and Chazelle [2] on the randomized and deterministic complexity of the MST problem.

Theorem 3 [9] *The minimum spanning forest of a graph with m edges can be computed by a randomized algorithm in O(m) time with probability* $1 - 2^{-\Omega(m)}$.

Theorem 4 [2] *The minimum spanning tree of a graph can be computed in* $O(m\alpha(m, n))$ *time by a deterministic algorithm, where* α *is the inverse-Ackermann function.*

Applications

Borůvka [1] invented the MST problem while considering the practical problem of electrifying rural Moravia (present day Czech Republic) with the shortest electrical network. MSTs are used as a starting point for heuristic approximations to the optimal traveling salesman tour and optimal Steiner tree, as well as other network design problems. MSTs are a component in other graph optimization algorithms, notably the single-source shortest path algorithms of Thorup [19] and Pettie–Ramachandran [15]. MSTs are used as a tool for visualizing data that is presumed to have a tree structure; for example, if a matrix contains dissimilarity data for a set of species, the minimum spanning tree of the associated graph will presumably group closely related species; see [7]. Other modern uses of MSTs include modeling physical systems [17] and image segmentation [8]; see [4] for more applications.

Open Problems

The chief open problem is to determine the *deterministic* complexity of the minimum spanning tree problem. By Theorem 1 this is tantamount to determining the decision-tree complexity of the MST problem.

Experimental Results

Moret and Shapiro [11] evaluated the performance of greedy MST algorithms using a variety of priority queues. They concluded that the best MST algorithm is Jarník's [7] (also attributed to Prim and Dijkstra; see [3,7,12]) as implemented with a pairing heap [13]. Katriel, Sanders, and Träff [10] designed and implemented a non-greedy randomized MST algorithm based on that of Karger et al. [9]. They concluded that on moderately dense graphs it runs substantially faster than the greedy algorithms tested by Moret and Shapiro.

Cross References

► Randomized Minimum Spanning Tree

Recommended Reading

1. Borůvka, O.: O jistém problému minimálním. Práce Moravské Přírodovědecké Společnosti **3**, 37–58 (1926). In Czech
2. Chazelle, B.: A minimum spanning tree algorithm with inverse-Ackermann type complexity. J. ACM **47**(6), 1028–1047 (2000)
3. Cormen, TH., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms. MIT Press, Cambridge (2001)
4. Eppstein, D.: Geometry in action: minimum spanning trees. http://www.ics.uci.edu/~eppstein/gina/mst.html
5. Gabow, H.N., Galil, Z., Spencer, T.H., Tarjan, R.E.: Efficient algorithms for finding minimum spanning trees in undirected and directed graphs. Combinatorica **6**, 109–122 (1986)
6. Garey, M.R., Johnson, D.S.: Computers and intractability: a guide to NP-completeness. Freeman, San Francisco (1979)
7. Graham, R.L., Hell, P.: On the history of the minimum spanning tree problem. Ann. Hist. Comput. **7**(1), 43–57 (1985)
8. Ion, A., Kropatsch, W.G., Haxhimusa, Y.: Considerations regarding the minimum spanning tree pyramid segmentation method. In: Proc. 11th Workshop Structural, Syntactic, and Statistical Pattern Recognition (SSPR). LNCS, vol. 4109, pp. 182–190. Springer, Berlin (2006)
9. Karger, D.R., Klein, P.N., Tarjan, R.E.: A randomized linear-time algorithm for finding minimum spanning trees. J. ACM **42**, 321–329 (1995)
10. Katriel, I., Sanders, P., Träff, J.L.: A practical minimum spanning tree algorithm using the cycle property. In: Proc. 11th Annual European Symposium on Algorithms. LNCS, vol. 2832, pp. 679–690. Springer, Berlin (2003)
11. Moret, B.M.E., Shapiro, H.D.: An empirical assessment of algorithms for constructing a minimum spanning tree. In: DIMACS Ser. Discrete Math. Theoret. Comput. Sci., vol. 15, Am. Math. Soc., Providence, RI (1994)
12. Pettie, S.: On the shortest path and minimum spanning tree problems. Ph.D. thesis, The University of Texas, Austin, August 2003
13. Pettie, S.: Towards a final analysis of pairing heaps. In: Proc. 46th Annual Symposium on Foundations of Computer Science (FOCS), 2005, pp. 174–183
14. Pettie, S., Ramachandran, V.: An optimal minimum spanning tree algorithm. J. ACM **49**(1), 16–34 (2002)
15. Pettie, S., Ramachandran, V.: A shortest path algorithm for real-weighted undirected graphs. SIAM J. Comput. **34**(6), 1398–1431 (2005)
16. Preparata, F.P., Shamos, M.I.: Computational geometry. Springer, New York (1985)
17. Subramaniam, S., Pope, S.B.: A mixing model for turbulent reactive flows based on euclidean minimum spanning trees. Combust. Flame **115**(4), 487–514 (1998)

18. Tarjan, R.E.: Data structures and network algorithms. In: CBMS-NSF Reg. Conf. Ser. Appl. Math., vol. 44. SIAM, Philadelphia (1983)
19. Thorup, M.: Undirected single-source shortest paths with positive integer weights in linear time. J. ACM **46**(3), 362–394 (1999)

Minimum Weighted Completion Time
1999; Afrati et al.

V.S. ANIL KUMAR[1], MADHAV V. MARATHE[2], SRINIVASAN PARTHASARATHY[3], ARAVIND SRINIVASAN[4]
[1] Network Dynamics and Simulation Science Laboratory, Bioinformatics Institute, Virginia Tech, Blacksburg, VA, USA
[2] Department of Computer Science and Virginia Bioinformatics Institute, Virginia Tech, Blacksburg, VA, USA
[3] IBM T.J. Watson Research Center, Hawthorne, NY, USA
[4] Department of Computer Science, University of Maryland, College Park, MD, USA

Keywords and Synonyms

Average weighted completion time

Problem Definition

The minimum weighted completion time problem involves (i) a set J of n jobs, a positive weight w_j for each job $j \in J$, and a release date r_j before which it cannot be scheduled; (ii) a set of m machines, each of which can process at most one job at any time; and (iii) an arbitrary set of positive values $\{p_{i,j}\}$, where $p_{i,j}$ denotes the time to process job j on machine i. A schedule involves assigning jobs to machines and choosing an order in which they are processed. Let C_j denote the completion time of job j for a given schedule. The *weighted completion time* of a schedule is defined as $\sum_{j \in J} w_j C_j$, and the goal is to compute a schedule that has the minimum weighted completion time.

In the scheduling notation introduced by Graham et al. [7], a scheduling problem is denoted by a 3-tuple $\alpha|\beta|\gamma$, where α denotes the machine environment, β denotes the additional constraints on jobs, and γ denotes the objective function. In this article, we will be concerned with the α-values 1, P, R, and Rm, which respectively denote one machine, identical parallel machines (i. e., for a fixed job j and for each machine i, $p_{i,j}$ equals a value p_j that is independent of i), unrelated machines (the $p_{i,j}$'s are dependent on both job i and machine j), and a fixed number m (not part of the input) of unrelated machines. The field β takes on the values r_j, which indicates that the jobs have release dates, and the value *pmtn*, which indicates that preemption of jobs is permitted. Further, the value *prec* in the field β indicates that the problem may involve precedence constraints between jobs, which poses further restrictions on the schedule. The field γ is either $\sum w_j C_j$ or $\sum C_j$, which denote total weighted and total (unweighted) completion times, respectively.

Some of the simpler classes of the weighted completion time scheduling problems admit optimal polynomial-time solutions. They include the problem $P||\sum C_j$, for which the *shortest-job-first* strategy is optimal, the problem $1||\sum w_j C_j$, for which Smith's rule [13] (scheduling jobs in their nondecreasing order of p_j/w_j values) is optimal, and the problem $R||\sum C_j$, which can be solved via matching techniques [2,9]. With the introduction of release dates, even the simplest classes of the weighted completion time minimization problem becomes strongly nondeterministic polynomial-time (NP)-hard. In this article, we focus on the work of Afrati et al. [1], whose main contribution is the design of polynomial-time approximation schemes (PTASs) for several classes of scheduling problems to minimize weighted completion time *with release dates*. Prior to this work, the best solutions for minimizing weighted completion time with release dates were all $O(1)$-approximation algorithms (e. g., [4,5,11]); the only known PTAS for a strongly NP-hard problem involving weighted completion time was due to Skutella and Woeginger [12], who developed a PTAS for the problem $P||\sum w_j C_j$. For an excellent survey on the minimum weighted completion time problem, we refer the reader to Chekuri and Khanna [3].

Key Results

Afrati et al. [1] were the first to develop PTASs for weighted completion time problems involving release dates. We summarize the running times of these PTASs in Table 1.

The results presented in Table 1 were obtained through a careful sequence of input transformations followed by dynamic programming. The input transformations ensure that the input becomes *well structured* at a slight loss in optimality, while dynamic programming allows efficient enumeration of all the near-optimal solutions to the well-structured instance.

The first step in the input transformation is *geometric rounding*, in which the processing times and release dates are converted to powers of $1 + \epsilon$, with at most $1 + \epsilon$ loss in the overall performance. More significantly, this step

Minimum Weighted Completion Time, Table 1
Summary of results of Afrati et al. [1]

Problem	Running time of polynomial-time approximation schemes
$1\|r_j\|\sum w_jC_j$	$O(2^{poly(\frac{1}{\epsilon})}n + n\log n)$
$P\|r_j\|\sum w_jC_j$	$O((m+1)^{poly(\frac{1}{\epsilon})}n + n\log n)$
$P\|r_j, pmtn\|\sum w_jC_j$	$O(2^{poly(\frac{1}{\epsilon})}n + n\log n)$
$Rm\|r_j\|\sum w_jC_j$	$O(f(m, \frac{1}{\epsilon})poly(n))$
$Rm\|r_j, pmtn\|\sum w_jC_j$	$O(f(m, \frac{1}{\epsilon})n + n\log n)$
$Rm\|\|\sum w_jC_j$	$O(f(m, \frac{1}{\epsilon})n + n\log n)$

(i) ensures that there are only a small number of distinct processing times and release dates to deal with, (ii) allows time to be broken into geometrically increasing intervals, and (iii) aligns release dates with start and end times of intervals. These are useful properties that can be exploited by dynamic programming.

The second step in the input transformation is *time stretching*, in which small amounts of idle time are added throughout the schedule. This step also changes completion times by a factor of at most $1 + O(\epsilon)$, but is useful for *cleaning up* the scheduling. Specifically, if a job is *large* (i. e., occupies a large portion of the interval where it executes), it can be pushed into the idle time of a later interval where it is small. This ensures that most jobs have small sizes compared with the length of the intervals where they execute, which greatly simplifies schedule computation. The next step is *job shifting*. Consider a partition of the time interval $[0, \infty)$ into intervals of the form $I_x = [(1+\epsilon)^x, (1+\epsilon)^{x+1})$, for integral values of x. The job-shifting step ensures that there is a slightly suboptimal schedule in which every job j gets completed within $O(\log_{1+\epsilon}(1 + \frac{1}{\epsilon}))$ intervals after r_j. This has the following nice property: If we consider blocks of intervals $\mathcal{B}_0, \mathcal{B}_1, \ldots$, with each block $\mathcal{B}_i$ containing $O(\log_{1+\epsilon}(1+\frac{1}{\epsilon}))$ consecutive intervals, then a job j starting in block $\mathcal{B}_i$ completes within the next block. Further, the other steps in the job-shifting phase ensure that there are not too many *large* jobs which spill over to the next block; this allows the dynamic programming to be done efficiently.

The precise steps in the algorithms and their analysis are subtle, and the above description is clearly an oversimplification. We refer the reader to [1] or [3] for further details.

Applications

A number of optimization problems in parallel computing and operations research can be formulated as machine scheduling problems. When precedence constraints are introduced between jobs, the weighted completion time objective can generalize the more commonly studied makespan objective, and hence is important.

Open Problems

Some of the major open problems in this area are to improve the approximation ratios for scheduling on unrelated or related machines for jobs with precedence constraints. The following problems in particular merit special mention. The best known solution for the $1|prec|\sum w_jC_j$ problem is the 2-approximation algorithm due to Hall et al. [8]; improving upon this factor is a major open problem in scheduling theory. The problem $R|prec|\sum_j w_jC_j$ in which the precedence constraints form an arbitrary acyclic graph is especially open – the only known results in this direction are when the precedence constraints form chains [6] or trees [10].

The other open direction is inapproximability – there are significant gaps between the known approximation guarantees and hardness factors for various problem classes. For instance, the $R||\sum w_jC_j$ and $R|r_j|\sum w_jC_j$ are both known to be approximable-hard, but the best known algorithms for these problems (due to Skutella [11]) have approximation ratios of 3/2 and 2 respectively. Closing these gaps remain a significant challenge.

Cross References

- ▶ Flow Time Minimization
- ▶ List Scheduling
- ▶ Minimum Flow Time
- ▶ Minimum Makespan on Unrelated Machines

Acknowledgements

The Research of V.S. Anil Kumar and M.V. Marathe was supported in part by NSF Award CNS-0626964. A. Srinivasan's research was supported in part by NSF Award CNS-0626636.

Recommended Reading

1. Afrati, F.N., Bampis, E., Chekuri, C., Karger, D.R., Kenyon, C., Khanna, S., Milis, I., Queyranne, M., Skutella, M., Stein, C., Sviridenko, M.: Approximation schemes for minimizing average weighted completion time with release dates. In: Proc. of Foundations of Computer Science, pp. 32–44 (1999)
2. Bruno, J.L., Coffman, E.G., Sethi, R.: Scheduling independent tasks to reduce mean finishing time. Commun. ACM **17**, 382–387 (1974)
3. Chekuri, C., Khanna, S.: Approximation algorithms for minimizing weighted completion time. In: J. Y-T. Leung (eds.) Handbook of Scheduling: Algorithms, Models, and Performance Analysis. CRC Press, Boca Raton (2004)

4. Chekuri, C., Motwani, R., Natarajan, B., Stein, C.: Approximation techniques for average completion time scheduling. SIAM J. Comput. **31**(1), 146–166 (2001)
5. Goemans, M., Queyranne, M., Schulz, A., Skutella, M., Wang, Y.: Single machine scheduling with release dates. SIAM J. Discret. Math. **15**, 165–192 (2002)
6. Goldberg, L.A., Paterson, M., Srinivasan, A., Sweedyk, E.: Better approximation guarantees for job-shop scheduling. SIAM J. Discret. Math. **14**, 67–92 (2001)
7. Graham, R.L., Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G.: Optimization and approximation in deterministic sequencing and scheduling: a survey. Ann. Discret. Math. **5**, 287–326 (1979)
8. Hall, L.A., Schulz, A.S., Shmoys, D.B., Wein, J.: Scheduling to minimize average completion time: off-line and on-line approximation algorithms. Math. Oper. Res. **22**(3), 513–544 (1997)
9. Horn, W.: Minimizing average flow time with parallel machines. Oper. Res. **21**, 846–847 (1973)
10. Kumar, V.S.A., Marathe, M.V., Parthasarathy, S., Srinivasan, A.: Scheduling on unrelated machines under tree-like precedence constraints. In: APPROX-RANDOM, pp. 146–157 (2005)
11. Skutella, M.: Convex quadratic and semidefinite relaxations in scheduling. J. ACM **46**(2), 206–242 (2001)
12. Skutella, M., Woeginger, G.J.: A PTAS for minimizing the weighted sum of job completion times on parallel machines. In: Proc. of 31st Annual ACM Symposium on Theory of Computing (STOC '99), pp. 400–407 (1999)
13. Smith, W.E.: Various optimizers for single-stage production. Nav. Res. Log. Q. **3**, pp. 59–66 (1956)

Minimum Weight Triangulation

1998; Levcopoulos, Krznaric

CHRISTOS LEVCOPOULOS
Department of Computer Science, Lund University, Lund, Sweden

Keywords and Synonyms

Minimum length triangulation

Problem Definition

Given a set S of n points in the Euclidean plane, a *triangulation* T of S is a maximal set of non-intersecting straight-line segments whose endpoints are in S. The *weight* of T is defined as the total Euclidean length of all edges in T. A triangulation that achieves minimum weight is called a *minimum weight triangulation*, often abbreviated MWT, of S.

Key Results

Since there is a very large number of papers and results dealing with minimum weight triangulation, only relatively very few of them can be mentioned here.

Mulzer and Rote have shown that MWT in NP-hard [11]. Their proof of NP-completeness is not given explicitly; it relies on extensive calculations which they performed with a computer. Also recently, Remy and Steger have shown a quasi-polynomial time approximation scheme for MWT [12]. These results are stated in the following theorem.

Theorem 1 *The problem of computing the MWT (minimum weight triangulation) of an input set S of n points in the plane is NP-hard. However, for any constant* $\epsilon > 0$, *a triangulation of S achieving the approximation ratio of* $1 + \epsilon$, *for an arbitrarily small positive constant* ϵ, *can be computed in time* $n^{\mathcal{O}(\log^8 n)}$.

The Quasi-Greedy Triangulation Approximates the MWT

Levcopoulos and Krznaric showed that a triangulation of total length within a constant factor of MWT can be computed in polynomial time for arbitrary point sets [7]. The triangulation achieving this result is a modification of the so-called *greedy* triangulation. The greedy triangulation starts with the empty set of diagonals and keeps adding a shortest diagonal not intersecting the diagonals which have already been added, until a full triangulation is produced. The greedy triangulation has been shown to approximate the minimum weight triangulation within a constant factor, unless a special case arises where the greedy diagonals inserted are "climbing" in a special, very unbalanced way along a relatively long concave chain containing many vertices and with a large empty space in front of it, at the same time blocking visibility from another, opposite concave chain of many vertices. In such "bad" cases the worst case ratio between the length of the greedy and the length of the minimum weight triangulation is shown to be $\Theta(\sqrt{n})$. To obtain a triangulation which always approximates the MWT within a constant factor, it suffices to take care of this special bad case in order to avoid the unbalanced "climbing", and replace it by a more balanced climbing along these two opposite chains. Each edge inserted in this modified method is still almost as short as the shortest diagonal, within a factor smaller than 1.2. Therefore, the modified triangulation which always approximates the MWT is named the *quasi-greedy* triangulation. In a similar way as the original greedy triangulation, the quasi-greedy triangulation can be computed in time $O(n \log n)$ [8]. Gudmundsson and Levcopoulos [5] showed later that a variant of this method can also be parallelized, thus achieving a constant factor approximation of MWT in $O(\log n)$ time, using $O(n)$ processors in the

CRCW PRAM model. Another by-product of the quasi-greedy triangulation is that one can easily select in linear time a subset of its edges to obtain a convex partition which is within a constant factor of the minimum length convex partition of the input point set. This last property was crucial in the proof that the quasi-greedy triangulation approximates the MWT. The proof also uses an older result that the (original, unmodified) greedy triangulation of any convex polygon approximates the minimum weight triangulation [9]. Some of the results from [7] and from [8] can be summarized in the following theorem:

Theorem 2 *Let S be an input set of n points in the plane. The quasi-greedy triangulation of S, which is a slightly modified version of the greedy triangulation of S, has total length within a constant factor of the length of the MWT (minimum weight triangulation) of S, and can be computed in time* $O(n \log n)$. *Moreover, the (unmodified) greedy triangulation of S has length within* $O(\sqrt{n})$ *of the length of MWT of S, and this bound is asymptotically tight in the worst case.*

Computing the Exact Minimum Weight Triangulation

Below three approaches to compute the exact MWT are shortly discussed. These approaches assume that it is numerically possible to efficiently compare the total length of sets of line segments in order to select the set of smallest weight. This is a simplifying assumption, since this is an open problem per se. However, the problem of computing the exact MWT remains NP-hard even under this assumption [11]. The three approaches differ with respect to the creation and selection of subproblems, which are then solved by dynamic programming.

The first approach, sketched by Lingas [10], employs a general method for computing optimal subgraphs of the complete Euclidean graph. By developing this approach, it is possible to achieve subexponential time $2^{O(\sqrt{n} \log n)}$. The idea to create the subproblems which are solved by dynamic programming. This is done by trying all (suitable) planar separators of length $O(\sqrt{n})$, separating the input point set in a balanced way, and then to proceed recursively within the resulting subproblems.

The second approach uses fixed-parameter algorithms. So, for example, if there are only $O(\log n)$ points in the interior of the convex hull of S, then the MWT of S can be computed in polynomial time [4]. This approach extends also to compute the minimum weight triangulation under the constraint that the outer boundary is not necessarily the convex hull of the input vertices, it can be an arbitrary polygon. Some of these algorithms have been implemented, see Grantson et al. [2] for a comparison of some implementations. These dynamic programming approaches take typically cubic time with respect to the points of the boundary, but exponential time with respect to the number of remaining points. So, for example, if k is the number of hole points inside the boundary polygon, then an algorithm, which has also been implemented, can compute the exact MWT in time $O(n^3 \cdot 2^k \cdot k)$ [2].

In an attempt to solve larger problems, a different approach uses properties of MWT which usually help to identify, for random point sets, many edges that *must* be, respectively *cannot* be, in MWT. One can then use dynamic programming to fill in the remaining MWT-edges. For random sets consisting of tens of thousands of points from the uniform distribution, one can thus compute the exact MWT in minutes [1].

Applications

The problem of computing a triangulation arises, for example, in finite element analysis, terrain modeling, stock cutting and numerical approximation [3,6]. The *minimum weight* triangulation has attracted the attention of many researchers, mainly due to its natural definition of optimality, and because it has proved to be a challenging problem over the past thirty years, with unknown complexity status until the end of 2005.

Open Problems

All results mentioned leave open problems. For example, can one find a simpler proof of NP-completeness, which can be checked without running computer programs? It would be desirable to improve the approximation constant which can be achieved in polynomial time (to simplify the proof, the constant shown in [7] is not explicitly calculated and it would be relatively large, if the proof is not refined). The time bound for the approximation scheme could hopefully be improved. It could also be possible to refine the software which computes efficiently the exact MWT for large random point sets, so that it can handle efficiently a wider range of input, i. e., not only completely random point sets. This could perhaps be done by combining this software with implementations of fixed parameter algorithms, as the ones reported in [2,4], or with other approaches. It is also open whether or not the subexponential exact method can be further improved.

Experimental Results

Please see the last paragraph under the section about key results.

URL to Code

Link to code used to compare some dynamic programming approaches in [2]: http://fuzzy.cs.uni-magdeburg.de/~borgelt/pointgon.html

Cross References

► Fast Minimal Triangulation
► Greedy Set-Cover Algorithms
► Minimum Geometric Spanning Trees
► Minimum k-Connected Geometric Networks

Recommended Reading

1. Beirouti, R., Snoeyink, J.: Implementations of the LMT Heuristic for Minimum Weight Triangulation. Symposium on Computational Geometry, pp. 96–105, Minneapolis, Minnesota, June 7–10, 1998
2. Borgelt, C., Grantson, M., Levcopoulos, C.: Fixed-Parameter Algorithms for the Minimum Weight Triangulation Problem. Technical Report LU-CS-TR:2006-238, ISSN 1650-1276 Report 158. Lund University, Lund (An extended version has been submitted to IJCGA) (2006)
3. de Berg, M., van Kreveld, M., Overmars, M., Schwarzkopf, O.: Computational Geometry – Algorithms and Applications, 2nd edn. Springer, Heidelberg (2000)
4. Grantson, M., Borgelt, C., Levcopoulos, C.: Minimum Weight Triangulation by Cutting Out Triangles. In: Proceedings 16th Annual International Symposium on Algorithms and Computation, ISAAC 2005, Sanya, China, pp. 984–994. Lecture Notes in Computer Science, vol. 3827. Springer, Heidelberg (2005)
5. Gudmundsson, J., Levcopoulos, C.: A Parallel Approximation Algorithm for Minimum Weight Triangulation. Nordic J. Comput. **7**(1), 32–57 (2000)
6. Hjelle, Ø., Dæhlen, M.: Triangulations and Applications. In: Mathematics and Visualization, vol. IX. Springer, Heidelberg (2006). ISBN 978-3-540-33260-2
7. Levcopoulos, C., Krznaric, D.: Quasi-Greedy Triangulations Approximating the Minimum Weight Triangulation. J. Algorithms **27**(2), 303–338 (1998)
8. Levcopoulos, C., Krznaric, D.: The Greedy Griangulation can be Computed from the Delaunay Triangulation in Linear Time. Comput. Geom. **14**(4), 197–220 (1999)
9. Levcopoulos, C., Lingas, A.: On Approximation Behavior of the Greedy Triangulation for Convex Polygons. Algorithmica **2**, 15–193 (1987)
10. Lingas, A.: Subexponential-time algorithms for minimum weight triangulations and related problems. In: Proceedings 10th Canadian Conference on Computational Geometry (CCCG), McGill University, Montreal, Quebec, 10–12 August 1998
11. Mulzer, W., Rote, G.: Minimum-weight triangulation is NP-hard. In: Proceedings 22nd Annual ACM Symposium on Computational Geometry, SoCG'06, Sedona, AZ, USA. ACM Press, New York, NY, USA (2006)
12. Remy, J., Steger, A.: A Quasi-Polynomial Time Approximation Scheme for Minimum Weight Triangulation. In: Proceedings 38th ACM Symposium on Theory of Computing (STOC'06). ACM Press, New York, NY, USA (2006)

Mobile Agents and Exploration

1952; Shannon

EVANGELOS KRANAKIS[1], DANNY KRIZANC[2]
[1] Department of Computer Science, Carleton, Ottawa, ON, Canada
[2] Department of Computer Science, Wesleyan University, Middletown, CT, USA

Keywords and Synonyms

Distributed algorithms; Graph exploration; Mobile agent; Navigation; Rendezvous; Routing; Time/Memory tradeoffs

Problem Definition

How can a network be explored efficiently with the help of mobile agents? This is a very broad question and to answer it adequately it will be necessary to understand more precisely what mobile agents are, what kind of networked environment they need to probe, and what complexity measures are interesting to analyze.

Mobile Agents

Mobile agents are autonomous, intelligent computer software that can move within a network. They are modeled as automata with limited memory and computation capability and are usually employed by another entity (to which they must report their findings) for the purpose of collecting information. The actions executed by the mobile agents can be discrete or continuous and transitions from one state to the next can be either deterministic or non-deterministic, thus giving rise to various natural complexity measures depending on the assumptions being considered.

Network Model

The network model is inherited directly from the theory of distributed computing. It is a connected graph whose vertices comprise the computing nodes and edges correspond to communication links. It may be static or dynamic and its resources may have various levels of accessibility. Depending on the model being considered, nodes and links of the network may have distinct labels. A particularly useful abstraction is an anonymous network whereby the nodes have no identities, which means that an agent cannot distinguish two nodes except perhaps by their degree. The outgoing edges of a node are usually thought of as distinguishable but an important distinction can be made between a globally consistent edge-labeling versus a locally independent edge-labeling.

Efficiency Measures for Exploration

Efficiency measures being adopted involve the time required for completing the exploration task, usually measured either by the number of edge traversals or nodes visited by the mobile agent. The interplay between time required for exploration and memory used by the mobile agent (*time/memory tradeoffs*) are key parameters considered for evaluating algorithms. Several researchers impose no restrictions on the memory but rather seek algorithms minimizing exploration time. Others, investigate the minimum size of memory which allows for exploration of a given type of network (e. g., tree) of given (known or unknown) size, regardless of the exploration time. Finally, several researchers consider time/memory tradeoffs.

Main Problems

Given a model for both the agents and the network, the graph exploration problem is that of designing an algorithm for the agent that allows it to visit all of the nodes and/or edges of the network. A closely related problem is where the domain to be explored is presented as a region of the plane with obstacles and exploration becomes visiting all unobstructed portions of the region in the sense of visibility. Another related problem is that of rendezvous where two or more agents are required to gather at a single node of a network.

Key Results

Claude Shannon [17] is credited with the first finite automaton algorithm capable of exploring an arbitrary maze (which has a range of 5×5 squares) by trial and error means. Exploration problems for mobile agents have been extensively studied in the scientific literature and the reader will find a useful historical introduction in Fraigniaud et al.[11].

Exploration in General Graphs

The network is modeled as a graph and the agent can move from node to node only along the edges. The graph setting can be further specified in two different ways. In Deng and Papadimitriou [8] the agent explores strongly connected directed graphs and it can move only in the direction from head to tail of an edge, but not vice-versa. At each point, the agent has a map of all nodes and edges visited and can recognize if it sees them again. They minimize the ratio of the total number of edges traversed divided by the optimum number of traversals, had the agent known the graph. In Panaite and Pelc [15] the explored graph is undirected and the agent can traverse edges in both directions. In the graph setting it is often required that apart from completing exploration the agent has to draw a map of the graph, i. e., output an isomorphic copy of it. Exploration of directed graphs assuming the existence of labels is investigated in Albers and Henzinger [1] and Deng and Papadimitriou [8]. Also in Panaite and Pelc [15], an exploration algorithm is proposed working in time $e + O(n)$, where is n the number of nodes and e the number of links. Fraigniaud et al. [11] investigate memory requirements for exploring unknown graphs (of unknown size) with unlabeled nodes and locally labeled edges at each node. In order to explore all graphs of diameter D and max degree d a mobile agent needs $\Omega(D \log d)$ memory bits even when exploration is restricted to planar graphs. Several researchers also investigate exploration of anonymous graphs in which agents are allowed to drop and remove pebbles. For example in Bender et al. [4] it is shown that one pebble is enough for exploration, if the agent knows an upper bound on the size of the graph, and $\Theta(\log \log n)$ pebbles are necessary and sufficient otherwise.

Exploration in Trees

In this setting it is assumed the agent can distinguish ports at a node (locally), but there is no global orientation of the edges and no markers available. *Exploration with stop* is when the mobile agent has to traverse all edges and stop at some node. For *exploration with return* the mobile agent has to traverse all edges and stop at the starting node. In *perpetual exploration* the mobile agent has to traverse all edges of the tree but is not required to stop. The upper and lower bounds on memory for the exploration algorithms analyzed in Diks et al. [9] are summarized in the table, depending on the knowledge that the mobile agent has. Here, n is the number of nodes of the tree, $N \geq n$ is an upper bound known to the mobile agent, and d is the maximum degree of a node of the tree.

Exploration	Knowledge	Lower Bounds	Upper Bounds
Perpetual	$\emptyset$	None	$O(\log d)$
w/Stop	$n \leq N$	$\Omega(\log \log \log n)$	$O(\log N)$
w/Return	$\emptyset$	$\Omega(\log n)$	$O(\log^2 n)$

Exploration in a Geometric Setting

Exploration in a geometric setting with unknown terrain and convex obstacles is considered by Blum et al. [5]. They compare the distance walked by the agent (or robot) to the length of the shortest (obstacle-free) path in the scene and describe and analyze robot strategies that minimize this ratio for different kinds of scenes. There is also related literature for exploration in more general settings with polygonal and rectangular obstacles by Deng et al. [7] and Bar-Eli et al. [3], respectively. A setting that is important in wireless networking is when nodes are aware of their location. In this case, Kranakis et al. [12] give efficient algorithms for navigation, namely compass routing and face routing that guarantee delivery in Delaunay and arbitrary planar geometric graphs, respectively, using only local information.

Rendezvous

The rendezvous search problem differs from the exploration problem in that it concerns two searchers placed at different nodes of a graph that want to minimize the time required to rendezvous (usually) at the same node. At any given time the mobile agents may occupy a vertex of the graph and can either stay still or move from vertex to vertex. It is of interest to minimize the time required to rendezvous. A natural extension of this problem is to study multi-agent mobile systems. More generally, given a particular agent model and network model, a set of agents distributed arbitrarily over the nodes of the network are said to rendezvous if executing their programs after some finite time they all occupy the same node of the network at the same time. Of special interest is the highly symmetric case of anonymous agents on an anonymous network and the simplest interesting case is that of two agents attempting to rendezvous on a ring network. In particular, in the model studied by Sawchuk [16] the agents cannot distinguish between the nodes, the computation proceeds in synchronous steps, and the edges of each node are oriented consistently. The table summarizes time/memory tradeoffs known for six algorithms (see Kranakis et al. [13] and Flocchini et al. [10]) when the k mobile agents use indistinguishable pebbles (one per mobile agent) to mark their position in an n node ring.

Memory	Time	Memory	Time
$O(k \log n)$	$O(n)$	$O(\log n)$	$O(n)$
$O(\log n)$	$O(kn)$	$O(\log k)$	$O(n)$
$O(k \log \log n)$	$O\left(\frac{n \log n}{\log \log n}\right)$	$O(\log k)$	$O(n \log k)$

Kranakis et al.[14] show a striking computational difference for rendezvous in an oriented, synchronous, $n \times n$ torus when the mobile agents may have more indistinguishable tokens. It is shown that two agents with a constant number of unmovable tokens, or with one movable token each cannot rendezvous if they have $o(\log n)$ memory, while they can perform rendezvous with detection as long as they have one unmovable token and $O(\log n)$ memory. In contrast, when two agents have two movable tokens each then rendezvous (respectively, rendezvous with detection) is possible with constant memory in a torus. Finally, two agents with three movable tokens each and constant memory can perform rendezvous with detection in a torus. If the condition on synchrony is dropped the rendezvous problem becomes very challenging. For a given initial location of agents in a graph, De Marco et al. [6] measure the performance of a rendezvous algorithm as the number of edge traversals of both agents until rendezvous is achieved. If the agents are initially situated at a distance D in an infinite line, they give a rendezvous algorithm with cost $O(D|L_{\min}|^2)$ when D is known and $O((D + |L_{\max}|)^3)$ if D is unknown, where $|L_{\min}|$ and $|L_{\max}|$ are the lengths of the shorter and longer label of the agents, respectively. These results still hold for the case of the ring of unknown size but then they also give an optimal algorithm of cost $O(n|L_{\min}|)$, if the size n of the ring is known, and of cost $O(n|L_{\max}|)$, if it is unknown. For arbitrary graphs, they show that rendezvous is feasible if an upper bound on the size of the graph is known and they give an optimal algorithm of cost $O(D|L_{\min}|)$ if the topology of the graph and the initial positions are known to the agents.

Applications

Interest in mobile agents has been fueled by two overriding concerns. First, to simplify the complexities of distributed computing, and second to overcome the limitations of user interface approaches. Today they find numerous applications in diverse fields such as distributed problem solving and planning (e. g., task sharing and coordination), network maintenance (e. g., daemons in networking systems for carrying out tasks like monitoring and surveillance), electronic commerce and intelligence search (e. g., data mining and surfing crawlers to find products and services from multiple sources), robotic exploration (e. g., rovers, and other mobile platforms that can explore potentially dangerous environments or even enhance planetary extravehicular activity), and distributed rational decision making (e. g., auction protocols, bargaining, decision making). The interested reader can find useful information in several articles in the volume edited by Weiss [18].

Open Problems

Specific directions for further research would include the study of time/memory tradeoffs in search game models (see Alpern and Gal [2]). Multi-agent systems are particularly useful for content-based searches and exploration, and further investigations in this area would be fruitful. Memory restricted mobile agents provide a rich model with applications in sensor systems. In the geometric setting, navigation and routing in a three dimensional environment using only local information is an area with many open problems.

Cross References

► Deterministic Searching on the Line
► Robotics
► Routing

Recommended Reading

1. Albers, S., Henzinger, M.R.: Exploring unknown environments. SIAM J. Comput. **29**, 1164–1188 (2000)
2. Alpern, S., Gal, S.: The Theory of Search Games and Rendezvous. Kluwer Academic Publishers, Norwell (2003)
3. Bar-Eli, E., Berman, P., Fiat, A., Yan, R.: On-line navigation in a room. J. Algorithms **17**, 319–341 (1994)
4. Bender, M.A., Fernandez, A., Ron, D., Sahai, A., Vadhan, S.: The power of a pebble: Exploring and mapping directed graphs. In: Proc. 30th Ann. Symp. on Theory of Computing, pp. 269–278. Dallas, 23–26 May 1998
5. Blum, A., Raghavan, P., Schieber, B.: Navigating in unfamiliar geometric terrain. SIAM J. Comput. **26**, 110–137 (1997)
6. De Marco, G., Gargano, L., Kranakis, E., Krizanc, D., Pelc, A., Vaccaro, U.: Asynchronous Deterministic Rendezvous in Graphs. Theoret. Comput. Sci. **355**, 315–326 (2006)
7. Deng, X., Kameda, T., Papadimitriou, C.H.: How to learn an unknown environment I: the rectilinear case. J. ACM **45**, 215–245 (1998)
8. Deng, X., Papadimitriou, C.H.: Exploring an unknown graph. J. Graph Theory **32**, 265–297 (1999)
9. Diks, K., Fraigniaud, P., Kranakis, E., Pelc, A.: Tree exploration with little memory. J. Algorithms **51**, 38–63 (2004)
10. Flocchini, P., Kranakis, E., Krizanc, D., Santoro, N., Sawchuk, C.: Multiple Mobile Agent Rendezvous in the Ring. In: Proc. LATIN 2004. LNCS, vol. 2976, pp. 599–608. Bueons Aires, 5–8 April 2004
11. Fraigniaud, P., Ilcinkas, D., Peer, G., Pelc, A., Peleg, D.: Graph exploration by a finite automaton. Theor. Comput. Sci. **345**, 331–344 (2005)
12. Kranakis, E., Singh, H., Urrutia, J.: Compass Routing in Geometric Graphs. In: Proceedings of 11th Canadian Conference on Computational Geometry, CCCG-99, pp. 51–54, Vancouver, 15–18 August 1999
13. Kranakis, E., Krizanc, D., Santoro, N., Sawchuk, C.: Mobile Agent Rendezvous Search Problem in the Ring. In: Proc. International Conference on Distributed Computing Systems (ICDCS), pp. 592–599. Providence, Rhode Island 19–22 May 2003
14. Kranakis, E., Krizanc, D., Markou, E.: Mobile Agent Rendezvous in a Synchronous Torus. In: Proceedings of LATIN 2006, 7th Latin American Symposium. Valdivia, March 20–24 2006. Correa, J., Hevia, A., Kiwi, M. SVLNCS **3887**, 653–664 (2006)
15. Panaite, P., Pelc, A.: Exploring unknown undirected graphs. J. Algorithms **33**, 281–295 (1999)
16. Sawchuk, C.: Mobile Agent Rendezvous in the Ring. Ph. D. thesis, Carleton University, Ottawa, Canada (2004)
17. Shannon, C.: Presentation of a Maze Solving Machine, in Cybernetics, Circular, Causal and Feedback Machines in Biological and Social Systems. In: von Feerster, H., Mead, M., Teuber, H.L. (eds.) Trans. 8th Conf, New York, March 15–16, 1951. pp. 169–181. Josiah Mary Jr. Foundation, New York (1952)
18. Weiss, G. (ed.): Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. MIT Press, Cambridge, MA (1999)

MST

► Minimum Energy Broadcasting in Wireless Networks
► Minimum Geometric Spanning Trees

Multicommodity Flow, Well-linked Terminals and Routing Problems

2005; Chekuri, Khanna, Shepherd

CHANDRA CHEKURI
Department of Computer Science, University of Illinois, Urbana-Champaign, Urbana, IL, USA

Keywords and Synonyms

Edge disjoint paths problem; Maximum edge disjoint paths problem; Node disjoint paths problem, All-or-nothing multicommodity flow problem

Problem Definition

Three related optimization problems derived from the classical edge disjoint paths problem (EDP) are described. An instance of EDP consists of an undirected graph $G = (V, E)$ and a multiset $\mathcal{T} = \{s_1 t_1, s_2 t_2, \dots, s_k t_k\}$ of k node pairs. EDP is a decision problem: can the pairs in $\mathcal{T}$ be connected (alternatively routed) via edge-disjoint paths? In other words, are there paths $P_1, P_2, \dots, P_k$ such that for $1 \le i \le k, P_i$ is path from s_i to t_i, and no edge $e \in E$ is in more than one of these paths? EDP is known to be NP-Complete. This article considers there maximization problems related to EDP.

- **Maximum Edge-Disjoint Paths Problem (MEDP).** Input to MEDP is the same as for EDP. The objective is to *maximize* the number of pairs in $\mathcal{T}$ that can be routed via edge-disjoint paths. The output consists of a subset $S \subseteq \{1, 2, \dots, k\}$ and for each $i \in S$ a path P_i connecting s_i to t_i such that the paths are edge-disjoint. The goal is to maximize $|S|$.

- **Maximum Edge-Disjoint Paths Problem with Congestion (MEDPwC).** MEDPwC is a relaxation of MEDP. The input, in addition to G and the node pairs, contains an integer congestion parameter c. The output is the same for MEDP; a subset $S \subseteq \{1, 2, \dots, k\}$ and for each $i \in S$ a path P_i connecting s_i to t_i. However, the paths $P_i, 1 \leq i \leq k$ are not required to be edge-disjoint. The relaxed requirement is that for each edge $e \in E$, the number of paths for the routed pairs that contain e is at most c. Note that MEDPwC with $c = 1$ is the same as MEDP.
- **All-or-Nothing Multicommodity Flow Problem (ANF).** ANF is a different relaxation of MEDP obtained by relaxing the notion of routing. A pair $s_i t_i$ is now said to be routed if a unit flow is sent from s_i to t_i (potentially on multiple paths). The input is the same as for MEDP. The output consists of a subset $S \subseteq \{1, 2, \dots, k\}$ such that there is a feasible multicommodity flow in G that routes one unit of flow for each pair in S. The goal is to maximize $|S|$.

In the rest of the article, graphs are assumed to be undirected multigraphs. Given a graph $G = (V, E)$ and $S \subset V$, let $\delta_G(S)$ denote the set of edges with exactly one end point in S. Let n denote the number of vertices in the input graph.

Key Results

A few results in the broader literature are reviewed in addition to the results from [6]. EDP is NP-Complete when k is part of the input. A highly non-trivial result of Robertson and Seymour yields a polynomial time algorithm when k is a fixed constant.

Theorem 1 ([16]) *There is a polynomial time algorithm for EDP when k is a fixed constant independent of the input size.*

Using Theorem 1 it is easy to see that MEDP and MEDPwC have polynomial time algorithms for fixed k . The same holds for ANF by simple enumeration since the decision version is polynomial-time solvable via linear programming.

The focus of this article is on the case when k is part of the input, and in this setting, all three problems considered are NP-hard. The starting point for most approximation algorithms is the natural multicommodity flow relaxation given below. This relaxation is valid for both MEDP and ANF. The end points of the input pairs are referred to as *terminals* and let X denote the set of terminals. To describe the relaxation as well as simplify further discussion, the following simple assumption is made without loss of generality; each node in the graph participates in at most one of the input pairs. This assumption implies that the input pairs induce a matching M on the terminal set X. Thus the input for the problem can alternatively given as a triple (G, X, M).

For the given instance (G, X, M), let $\mathcal{P}_i$ denote the set of paths joining s_i and t_i in G and let $\mathcal{P} = \cup_i \mathcal{P}_i$. The LP relaxation has the following variables. For each path $P \in \mathcal{P}$ there is a variable $f(P)$ which is the amount of flow sent on P. For each pair $s_i t_i$ there is a variable x_i to indicate the total flow that is routed for the pair.

$$
\begin{aligned}
(\mathrm{MCF-LP}) \max \sum_{i=1}^{k} x_i \quad \text{s.t} \\
x_i - \sum_{P \in \mathcal{P}_i} f(P) = 0 \quad 1 \leq i \leq k \\
\sum_{P:\, e \in P} f(P) \leq 1 \quad \forall e \in E \\
x_i, f(P) \in [0,1] \quad 1 \leq i \leq k, P \in \mathcal{P}
\end{aligned}
$$

The above path formulation has an exponential (in n) number of variables, however it can still be solved in polynomial time. There is also an equivalent compact formulation with a polynomial number of variables and constraints. Let OPT denote the value of an optimum solution to a given instance. Similarly, let OPT-LP denote the value of an optimum solution the LP relaxation for the given instance. It can be seen that OPT-LP $\geq$ OPT. It is known that the integrality gap of (MCF-LP) is $\Omega(\sqrt{n})$ [10]; that is, there is an infinite family of instances such that $OPT-LP/OPT = \Omega(\sqrt{n}$. The current best approximation algorithm for MEDP is given by the following theorem.

Theorem 2 ([4]) *The integrality gap of (MCF-LP) for MEDP is $\Theta(\sqrt{n})$ and there is an $O(\sqrt{n})$ approximation for MEDP.*

For MEDPwC the approximation ratio improves with the congestion parameter c.

Theorem 3 ([18])

There is an $O(n^{1/c})$ approximation for MEDPwC with congestion parameter c. In particular there is a polynomial time algorithm that routes $\Omega(\text{OPT-LP}/n^{1/c})$ pairs with congestion at most c.

The above theorem is established via randomized rounding of a solution to (MCF-LP). Similar results, but via simpler combinatorial algorithms, are obtained in [2,15].

In [6] a new framework was introduced to obtain approximation algorithm for routing problems in undirected graphs via (MCF-LP). A key part of the framework is the

Multicommodity Flow, Well-linked Terminals and Routing Problems, Table 1
Known bounds for MEDP, ANF and MEDPwC in general undirected graphs. The best upper bound on the approximation ratio is the same as the upper bound on the integrality gap of (MCF-LP)

	Integrality Gap of (MCF-LP)		*Approximation Ratio*
	Upper bound	*Lower bound*	*Lower bound*
MEDP	$O(\sqrt{n})$	$\Omega(\sqrt{n})$	$\Omega(\log^{1/2-\epsilon} n)$
MEDPwC	$O(n^{1/c})$	$\Omega(\log^{(1-\epsilon)/(c+1)} n)$	$\Omega(\log^{(1-\epsilon)/(c+1)} n)$
ANF	$O(\log^2 n)$	$\Omega(\log^{1/2-\epsilon} n)$	$\Omega(\log^{1/2-\epsilon} n)$

so-called well-linked decomposition that allows a reduction of an arbitrary instance to an instance in which the terminals satisfy a strong property.

Definition 1 Let $G = (V, E)$ be a graph. A subset $X \subseteq V$ is *cut-well-linked* in G if for every $S \subset V$, $|\delta_G(S)| \geq \min\{|S \cap X|, |(V \setminus S) \cap X|\}$. X is *flow-well-linked* if there exists a feasible fractional multicommodity flow in G for the instance in which there is a demand of $1/|X|$ for each unordered pair $uv, u, v \in X$.

The main result in [6] is the following.

Theorem 4 ([6]) *Let (G, X, M) be an instance of MEDP or ANF and let* $\textsc{opt-lp}$ *be the value of an optimum solution to (MCF-LP) on (G, X, M). There there is a polynomial time algorithm that obtains a collection of instances $(G_1, X_1, M_1), (G_2, X_2, M_2), \ldots, (G_h, X_h, M_h)$ with the following properties:*

- *The graphs $G_1, G_2, \ldots, G_h$ are node-disjoint induced subgraphs of G. For $1 \leq i \leq h$, $X_i \subseteq X$ and $M_i \subseteq M$.*
- *For $1 \leq i \leq h$, X_i is flow-well-linked in G_i.*
- $\sum_{i=1}^{h} |X_i| = \Omega(\textsc{opt-lp}/\log^2 n)$.

For planar graphs and graphs that exclude a fixed minor, the above theorem gives a stronger guarantee: $\sum_{i=1}^{h} |X_i| = \Omega(\textsc{opt-lp}/\log n)$. A well-linked instance satisfies a strong symmetry property based on the following observation. If A is flow-well-linked in G then for *any* matching J on X, $\textsc{opt-lp}$ on the instance (G, A, J) is $\Omega(|A|)$. Thus the particular matching M of a given well-linked instance (G, X, M) is essentially irrelevant. The second part of the framework in [6] consists of exploiting the well-linked property of the instances produced by the decomposition procedure. At a high level this is done by showing that if G has a well-linked set X, then it contains a "crossbar" (a routing structure) of size $\Omega(|X|/\text{poly}(\log n))$. See [6] for more precise definitions. Techniques for the second part vary based on the problem as well as the family of graphs in question. The following results are obtained using Theorem 4 and other non-trivial ideas for the second part [7,8,6,3].

Theorem 5 ([6]) *There is an $O(\log^2 n)$ approximation for ANF. This improves to an $O(\log n)$ approximation in planar graphs.*

Theorem 6 ([6]) *There is an $O(\log n)$ approximation for MEDPwC in planar graphs for $c \geq 2$. There is an $O(\log n)$ approximation for ANF in planar graphs.*

Theorem 7 ([3]) *There is an $O(r \log n \log r)$ approximation for MEDP in graphs of treewidth at most r.*

Generalizations and Variants

Some natural variants and generalizations of the problems mentioned in this article are obtained by considering three orthogonal aspects: (i) node disjointness instead of edge-disjointness, (ii) capacities on the edges and/or nodes, and (iii) demand values on the pairs (each pair $s_i t_i$ has an integer demand d_i and the objective is to route d_i units of flow between s_i and t_i). Results similar to those mentioned in the article are shown to hold for these generalizations and variants [6]. Capacities and demand values on pairs are somewhat easier to handle while node-disjoint problems often require additional non-trivial ideas. The reader is referred to [6] for more details.

For some special classes of graphs (trees, expanders and grids to name a few), constant factor or poly-logarithmic approximation ratios are known for MEDP.

Applications

Flow problems are at the core of combinatorial optimization and have numerous applications in optimization, computer science and operations research. Very special cases of EDP and MEDP include classical problems such as single-commodity flows, and matchings in general graphs, both of which have many applications. EDP and variants arise most directly in telecommunication networks and VLSI design. Since EDP captures difficult problems as special cases, there are only a few algorithmic tools that can address the numerous applications in a unified fashion. Consequently, empirical research tends to focus on application specific approaches to obtain satisfactory solutions.

The flip side of the difficulty of EDP is that it offers a rich source of problems, the study of which has led to important algorithmic advances of broad applicability, as well as fundamental insights in graph theory, combinatorial optimization, and related fields.

Open Problems

A number of very interesting open problems remain regarding the approximability of the problems discussed in this article. Table 1 gives the best known upper and lower bounds on the approximation ratio as well as integrality gap of (MCF-LP). All the inapproximability results in Table 1, and the integrality gap lower bounds for MEDPwC and ANF, are from [1]. The inapproximability results are based on the assumption that NP $\not\subseteq$ ZTIME($n^{\text{poly}(\log n)}$). Closing the gaps between the lower and upper bounds are the major open problems.

Cross References

► Randomized Rounding
► Separators in Graphs
► Treewidth of Graphs

Recommended Reading

The limited scope of this article does not do justice to the large literature on EDP and related problems. In addition to the articles cited in the main body of the article, the reader is referred to [5,9,11,12, 13,14,17] for further reading and pointers to existing literature.

1. Andrews, M., Chuzhoy, J., Khanna, S., Zhang, L.: Hardness of the Undirected Edge-Disjoint Paths Problem with Congestion. Proc. of IEEE FOCS, 2005, pp. 226–244
2. Azar, Y., Regev, O.: Combinatorial algorithms for the unsplittable flow problem. Algorithmica **44**(1), 49–66 (2006). Preliminary version in Proc. of IPCO 2001
3. Chekuri, C., Khanna, S., Shepherd, F.B.: A note on multiflows and treewidth. Algorithmica, published online (2007)
4. Chekuri, C., Khanna, S., Shepherd, F.B.: An $O(\sqrt{n})$ approximation and integrality gap for disjoint paths and UFP. Theor. Comput. **2**, 137–146 (2006)
5. Chekuri, C., Khanna, S., Shepherd, F.B.: Edge-Disjoint Paths in Planar Graphs with Constant Congestion. Proc. ACM STOC, pp. 757–766 (2006)
6. Chekuri, C., Khanna, S., Shepherd, F.B.: Multicommodity flow, well-linked terminals, and routing problems. Proc. ACM STOC, pp. 183–192 (2005)
7. Chekuri, C., Khanna, S., Shepherd, F.B.: The All-or-Nothing Multicommodity Flow Problem. Proc. ACM STOC, pp. 156–165 (2004)
8. Chekuri, C., Khanna, S., Shepherd, F.B.: Edge Disjoint Paths in Planar Graphs. Proc. of IEEE FOCS, 2004, pp. 71–80
9. Frank, A.: Packing paths, cuts, and circuits – a survey. In: Korte, B., Lovász, L., Prömel H.J., Schrijver A. (eds.) Paths, Flows and VLSI-Layout, pp. 49–100. Springer, Berlin (1990)
10. Garg, N., Vazirani, V., Yannakakis, M.: Primal-Dual Approximation Algorithms for Integral Flow and Multicut in Trees. Algorithmica **18**(1), 3–20 (1997). Preliminary version appeared in Proc. ICALP 1993
11. Guruswami, V., Khanna, S., Rajaraman, R., Shepherd, F.B., Yannakakis, M.: Near-Optimal Hardness Results and Approximation Algorithms for Edge-Disjoint Paths and Related Problems. J. CSS **67**, 473–496 (2003). Preliminary version in Proc. of ACM STOC 1999
12. Kleinberg, J.M.: Approximation algorithms for disjoint paths problems. Ph. D. thesis, MIT, Cambridge, MA (1996)
13. Kleinberg, J.M.: An Approximation Algorithm for the Disjoint Paths Problem in Even-Degree Planar Graphs. Proc. of IEEE FOCS, 2005, pp. 627–636
14. Kolliopoulos, S.G.: Edge Disjoint Paths and Unsplittable Flow. In: Handbook on Approximation Algorithms and Metaheuristics, Chapman & Hall/CRC Press Computer & Science Series, vol 13. Chapman Hall/CRC Press, May 2007
15. Kolliopoulos, S.G., Stein, C.: Approximating Disjoint-Path Problems Using Greedy Algorithms and Packing Integer Programs. Math. Program. A **99**, 63–87 (2004). Preliminary version in Proc. of IPCO 1998
16. Robertson, N., Seymour, P.D.: Graph Minors XIII.The Disjoint Paths Problem. J. Comb. Theor. B **63**(1), 65–110 (1995)
17. Schrijver, A.: Combinatorial Optimization: Polyhedra and Efficiency. Springer, Berlin (2003)
18. Srinivasan, A.: Improved approximations for edge-disjoint paths, unsplittable flow, and related routing problems. Proc. IEEE FOCS, 1997, pp. 416–425

Multicut

1993; Garg, Vazirani, Yannakakis
1996; Garg, Vazirani, Yannakakis

SHUCHI CHAWLA
Department of Computer Science,
University of Wisconsin–Madison, Madison, WI, USA

Problem Definition

The Multicut problem is a natural generalization of the s-t mincut problem—given an undirected capacitated graph $G = (V, E)$ with k pairs of vertices $\{s_i, t_i\}$; the goal is to find a subset of edges of the smallest total capacity whose removal from G disconnects s_i from t_i for every $i \in \{1, \cdots, k\}$. However, unlike the Mincut problem which is polynomial-time solvable, the Multicut problem is known to be NP-hard and APX-hard for $k \geq 3$ [6].

This problem is closely related to the Multi-Commodity Flow problem. The input to the latter is a capacitated network with k commodities (source-sink pairs); the goal is to route as much total flow between these source-sink pairs as possible while satisfying capacity constraints. The maximum multi-commodity flow in a graph can be found in polynomial time via linear programming, and there are also several combinatorial FPTASes known for this problem [7,9,11].

It is immediate from the definition of Multicut that the multicommodity flow in a graph is bounded above by the capacity of a minimum multicut in the graph. When there is a single commodity to be routed, the **max-flow min-cut** theorem of Ford and Fulkerson [8] states that the converse also holds: the maximum s-t flow in a graph is exactly equal to the minimum s-t cut in the graph. This duality between flows and cuts in a graph has many applications and, in particular, leads to a simple algorithm for finding the minimum cut in a graph.

Given its simplicity and elegance, several attempts have been made to extend this duality to other classes of flow and partitioning problems. Hu showed, for example, that the min-multicut equals the maximum multicommodity flow when there are only two commodities in the graph [12]. Unfortunately, this property does not extend to graphs with more than two commodities. The focus has therefore been on obtaining approximate max-multicommodity flow min-multicut theorems. Such theorems would also imply a polynomial-time algorithm for approximately computing the capacity of the minimum multicut in a graph.

Key Results

Garg, Vazirani and Yannakakis [10] were the first to obtain an approximate max-multicommodity flow min-multicut theorem. They showed that the maximum multicommodity flow in a graph is always at least an $O(\log k)$ fraction of the minimum multicut in the graph. Moreover, their proof of this result is constructive. That is, they also provide an algorithm for computing a multicut for a given graph with capacity at most $O(\log k)$ times the maximum multicommodity flow in the graph. This is the best approximation algorithm known to date for the Multicut problem.

Theorem 1 *Let M denote the minimum multicut in a graph with k commodities and f denote the maximum multicommodity flow in the graph. Then*

$$\frac{M}{O(\log k)} \le f \le M .$$

Moreover, there is a polynomial time algorithm for finding an $O(\log k)$-approximate multicut in a graph.

Furthermore, they show that this theorem is tight to within constant factors. That is, there are families of graphs in which the gap between the maximum multicommodity flow and minimum multicut is $\Theta(\log k)$.

Theorem 2 *There exists a infinite family of multicut instances $\{(G_k, P_k)\}$ such that for all k, the graph $G_k = (V_k, E_k)$ contains k vertices and $P_k \subseteq V_k \times V_k$ is a set of $\Omega(k^2)$ source-sink pairs. Furthermore, the maximum multicommodity flow in the instance (G_k, P_k) is $O(k/\log k)$ and the minimum multicut is $\Omega(k)$.*

Garg et al. also consider the Sparsest Cut problem which is another partitioning problem closely related to Multicut, and provided an approximation algorithm for this problem. Their results for Sparsest Cut have subsequently been improved upon [3,15]. The reader is referred to the entry on ► Sparsest Cut for more details.

Applications

A key application of the Multicut problem is to the 2CNF ≡Deletion problem. The latter is a constraint satisfaction problem in which given a weighted set of clauses of the form $P \equiv Q$, where P and Q are literals, the goal is to delete a minimum weight set of clauses so that the remaining set is satisfiable. The 2CNF ≡Deletion problem models a number of partitioning problems, for example the Minimum Edge-Deletion Graph Bipartization problem—finding the minimum weight set of edges whose deletion makes a graph bipartite. Klein et al. [14] showed that the 2CNF ≡Deletion problem reduces in an approximation preserving way to Multicut. Therefore, a ρ-approximation to Multicut implies a ρ-approximation to 2CNF ≡Deletion. (See the survey by Shmoys [16] for more applications.)

Open Problems

There is a big gap between the best-known algorithm for Multicut and the best hardness result (APX-hardness) known for the problem. Improvements in either direction may be possible, although there are indications that the $O(\log k)$ approximation is the best possible. In particular, Theorem 2 implies that the integrality gap of the natural linear programming relaxation for Multicut is $\Theta(\log k)$. Although improved approximations have been obtained for other partitioning problems using semi-definite programming instead of linear programming, Agarwal et al. [1] showed that similar improvements cannot be achieved for Multicut—the integrality gap of the natural SDP-relaxation for Multicut is also $\Theta(\log k)$. On the other hand, there are indications that the APX-hardness is not tight. In particular, assuming the so-called Unique Games conjecture, it has been shown that Multicut cannot be approximated to within any constant factor [4,13]. In light of these negative results, the main open problem related to this work is to obtain a super-constant hardness for the Multicut problem under a standard assumption such as $P \neq NP$.

The Multicut problem has also been studied in directed graphs. The best known approximation algorithm for this problem is an $O(n^{11/23} \log^{O(1)} n)$-approximation due to Aggarwal, Alon and Charikar [2], while on the hardness side, Chuzhoy and Khanna [5] show that there is no $2^{\Omega(\log^{1-\epsilon} n)}$ approximation, for any $\epsilon > 0$, unless NP$\subseteq$ZPP. Chuzhoy and Khanna also exhibit a family of instances for which the integrality gap of the natural LP relaxation of this problem (which is also the gap between the maximum directed multicommodity flow and the minimum directed multicut) is $\Omega(n^{1/7})$.

Cross References

► Sparsest Cut

Recommended Reading

1. Agarwal, A., Charikar, M., Makarychev, K., Makarychev, Y.: $O(\sqrt{\log n})$ approximation algorithms for Min UnCut, Min 2CNF Deletion, and directed cut problems. In: Proceedings of the 37th ACM Symposium on Theory of Computing (STOC), pp. 573–581, Baltimore, May 2005
2. Aggarwal, A., Alon, N., Charikar, M.: Improved approximations for directed cut problems. In: Proceedings of the 39th ACM Symposium on Theory of Computing (STOC), pp. 671–680, San Diego, June 2007
3. Arora, S., Satish, R., Vazirani, U.: Expander Flows, Geometric Embeddings, and Graph Partitionings. In: Proceedings of the 36th ACM Symposium on Theory of Computing (STOC), pp. 222–231, Chicago, June 2004
4. Chawla, S., Krauthgamer, R., Kumar, R., Rabani Y., Sivakumar, D.: On the Hardness of Approximating Sparsest Cut and Multicut. In: Proceedings of the 20th IEEE Conference on Computational Complexity (CCC), pp. 144–153, San Jose, June 2005
5. Chuzhoy, J., Khanna, S.: Polynomial flow-cut gaps and hardness of directed cut problems. In: Proceedings of the 39th ACM Symposium on Theory of Computing (STOC), pp. 179–188, San Diego, June 2007
6. Dahlhaus, E., Johnson, D.S., Papadimitriou, C.H., Seymour, P.D., Yannakakis, M.: The complexity of multiterminal cuts. SIAM Comput. J. **23**(4), 864–894 (1994)
7. Fleischer, L.: Approximating fractional multicommodity flow independent of the number of commodities. In: Proceedings of the 40th IEEE Symposium on Foundations of Computer Science (FOCS), pp. 24–31, New York, October 1999
8. Ford, L.R., Fulkerson, D.R.: Maximal flow through a network. Can. J. Math. **8**, 399–404. (1956)
9. Garg, N., Könemann., J.: Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In: Proceedings of the 39th IEEE Symposium on Foundations of Computer Science (FOCS), pp. 300–309. (1998)
10. Garg, N., Vazirani, V.V., Yannakakis, M.: Approximate max-flow min-(multi)cut theorems and their applications. SIAM Comput. J., **25**(2), 235–251. (1996)
11. Grigoriadis, M.D., Khachiyan, L.G.: Coordination complexity of parallel price-directive decomposition. Mathematics of Operations Research, **21**, 321–340. (1996)
12. Hu, T.C.: Multi-commodity network flows. Operations Research, **11**(3), 344–360. (1963)
13. Khot, S., Vishnoi, N.: The Unique Games Conjecture, Integrality Gap for Cut Problems and the Embeddability of Negative-Type Metrics into ℓ_1. In: Proceedings of the 46th IEEE Symposium on Foundations of Computer Science (FOCS), pp. 53–62. (2005)
14. Klein, P., Agrawal, A., Ravi, R., Rao, S.: Approximation through multicommodity flow. In: Proceedings of the 31st IEEE Symposium on Foundations of Computer Science (FOCS), pp. 726–737 (1990)
15. Linial, N., London, E., Rabinovich, Y.: The geometry of graphs and some of its algorithmic applications. Combinatorica **15**(2), 215–245 (1995). Also in Proc. 35th FOCS, pp. 577–591 (1994)
16. Shmoys, D.B.: Cut problems and their application to divide-and-conquer. In: Hochbaum D.S., (ed.), Approximation Algorithms for NP-hard Problems, pp. 192–235. PWS Publishing, Boston (1997)

Multidimensional Compressed Pattern Matching

2003; Amir, Landau, Sokol

AMIHOOD AMIR[1,2]

[1] Department of Computer Science, Bar-Ilan University, Ramat-Gan, Israel

[2] Department of Computer Science, John Hopkins University, Baltimore, MD, USA

Keywords and Synonyms

Pattern matching in compressed images; Two-dimensional compressed matching; Multidimensional compressed search

Problem Definition

Let c be a given compression algorithm, and let $c(D)$ be the result of c compressing data D. The *compressed search problem with compression algorithm c* is defined as follows.

INPUT: Compressed text $c(T)$ and pattern P.

OUTPUT: All locations in T where pattern P occurs.

A compressed matching algorithm is *optimal* if its time complexity is $O(|c(T)|)$.

Although optimality in terms of time is always important, when dealing with compression, the criterion of **extra**

space is perhaps more important [20]. Applications employ compression techniques specifically because there is a limited amount of available space. Thus, it is not sufficient for a compressed matching algorithm to be optimal in terms of time, it must also satisfy the given space constraints. Space constraints may be due to limited amount of disk space (e. g., on a server), or they may be related to the size of the memory or cache. Note that if an algorithm uses as little extra space as the size of the cache, the runtime of the algorithm is also greatly reduced as no cache misses will occur [13]. It is also important to remember that in many applications, e. g., LZ compression on strings, the *compression ratio* – $|S|/|c(S)|$ – is a small constant. In a case where the compression ratio of the given text is a constant, an optimal compressed matching performs no better than the naive algorithm of decompressing the text. However, if the constants hidden in the *"big O"* are smaller than the compression ratio, then the compressed matching does offer a practical benefit. If those constants are larger than the optimal the compressed search algorithm may, in fact, be using more space than the uncompressed text.

Definition 1 (inplace) A compressed matching is said to be *inplace* if the extra space used is proportional to the input size of the pattern.

Note that this definition encompasses the compressed matching model (e. g., [2]) where the pattern is input in uncompressed form, as well as the *fully compressed* model [10], where the pattern is input in compressed form. The *inplace* requirement allows the extra space to be the input size of the pattern, whatever that size may be. However, in many applications the compression ratio is a constant; therefore, a stronger space constraint is defined.

Definition 2 Let $\mathcal{AP}$ be the set of all patterns of size m, and let $c(\mathcal{AP})$ be the set of all compressed images of $\mathcal{AP}$. Let m' be the length of the smallest pattern in $c(\mathcal{AP})$. A compressed matching algorithm with input pattern P of length m is called *strongly inplace* if the amount of extra space used is proportional to m'.

The problem as defined above is equally applicable to textual (one-dimensional), image (two-dimensional), or any type of data, such as bitmaps, concordances, tables, XML data, or any possible data structure.

The compressed matching problem is considered crucial in image databases, since they are highly compressible. The initial definition of the compressed matching paradigm was motivated by the *two dimensional run-length compression*. This is the compression used for fax transmissions. The run-length compression is defined as follows.

Let $S = s_1 s_2 \cdots s_n$ be a string over some alphabet Σ. The *run-length compression* of string S is the string $S' = \sigma_1^{r_1} \sigma_2^{r_2} \cdots \sigma_k^{r_k}$ such that (1) $\sigma_i \neq \sigma_{i+1}$ for $1 \leq i < k$ and (2) S can be described as the concatenation of k *segments*, the symbol σ_1 repeated r_1 times, the symbol σ_2 repeated r_2 times, . . ., and the symbol σ_k repeated r_k times. The *two-dimensional run-length compression* is the concatenation of the run-length compression of all the matrix rows (or columns).

The *two-dimensional run-length compressed matching problem* is defined as follows:

INPUT: Text array T of size $n \times n$, and pattern array P of size $m \times m$ *both* in two-dimensional run-length compressed form.

OUTPUT: All locations in T of occurrences of P. Formally, the output is the set of locations (i,j) such that $T[i+k, j+l] = P[k+1, l+1]$ $k, l = 0 \dots m-1$.

Another ubiquitous lossless two-dimensional compression is CompuServe's GIF standard, widely used on the World Wide Web. It uses LZW [19] (a variation of LZ78) on the image linearized row by row.

The *two-dimensional LZ compression* is formally defined as follows. Given an image $T[1 \dots n, 1 \dots n]$, create a string $T_{lin}[1 \dots n^2]$ by concatenating all rows of T. Compressing T_{lin} with one-dimensional LZ78 yields the two-dimensional LZ compression of the image T.

The *two-dimensional LZ compressed matching problem* is defined as follows:

INPUT: Text array T of size $n \times n$, and pattern array P of size $m \times m$ *both* in two-dimensional LZ compressed form.

OUTPUT: All locations in T of occurrences of P. Formally, the output is the set of locations (i,j) such that $T[i+k, j+l] = P[k+1, l+1]$ $k, l = 0 \dots m-1$.

Key Results

The definition of compressed search first appeared in the context of searching for two dimensional run-length compression [1,2]. The following result was achieved there.

Theorem 1 (Amir and Benson [3]) *There exists an* $O(|c(T)| \log |c(T)|)$ *worst-case time solution to the compressed search problem with the two dimensional run-length compression algorithm.*

The abovementioned paper did not succeed in achieving either an optimal or an inplace algorithm. Nevertheless, it introduced the notion of *two-dimensional periodicity*. As in strings, periodicity plays a crucial rôle in two-dimensional string matching, and its advent has provided solutions to many longstanding open problems of two-dimensional string matching. In [5], it was used to achieve the first linear-time, alphabet-independent, two-dimensional text scanning. Later, in [4,16] it was used in two different ways for a linear-time witness table construction. In [7] it was used to achieve the first parallel, time and work optimal, CREW algorithm for text scanning. A simpler variant of periodicity was used by [11] to obtain a constant-time CRCW algorithm for text scanning. A recent further attempt has been made [17] to generalize periodicity analysis to higher dimensions.

The first optimal two-dimensional compressed search algorithm was the following.

Theorem 2 (Amir et al. [6]) *There exists an $O(|c(T)|)$ worst-case time solution to the compressed search problem with the two-dimensional run-length compression algorithm.*

Optimality was achieved by a concept the authors called *witness-free dueling*. The paper proved new properties of two-dimensional periodicity. This enables duels to be performed in which no witness is required. At the heart of the dueling idea lies the concept that two overlapping occurrences of a pattern in a text can use the content of a predetermined text position or witness in the overlap to eliminate one of them. Finding witnesses is a costly operation in a compressed text; thus, the importance of witness-free dueling.

The original algorithm of Amir et al. [6] takes time $O(|c(T)| + |P| \log \sigma)$, where σ is $\min(|P|, |\Sigma|)$, and Σ is the alphabet. However with the witness table construction of Galil and Park [12] the time is reduced to $O(|c(T)| + |P|)$. Using known techniques, one can modify their algorithm so that its extra space is $O(|P|)$. This creates an optimal algorithm that is also inplace, provided the pattern is input in uncompressed form. With use of the run-length compression, the difference between $|P|$ and $|c(P)|$ can be quadratic. Therefore it is important to seek an inplace algorithm.

Theorem 3 (Amir et al. [9]) *There exists an $O(|c(T)| + |P| \log \sigma)$ worst-case time solution to the compressed search problem with the two-dimensional run-length compression algorithm, where σ is $\min(|P|, |\Sigma|)$, and Σ is the alphabet, for all patterns that have no trivial rows (rows consisting of a single repeating symbol). The amount of space used is $O(|c(P)|)$.*

This algorithm uses the framework of the noncompressed two dimensional pattern matching algorithm of [6]. The idea is to use the **dueling** mechanism defined by Vishkin [18]. Applying the dueling paradigm directly to run-length compressed matching has previously been considered impossible since the location of a witness in the compressed text cannot be accessed in constant time. In [9], a way was shown in which a witness *can* be accessed in (amortized) constant time, enabling a relatively straightforward application of the dueling paradigm to compressed matching.

A strongly inplace compressed matching algorithm exists for the two-dimensional LZ compression, but its preprocessing is not optimal.

Theorem 4 (Amir et al. [8]) *There exists an $O(|c(T)| + |P|^3 \log \sigma)$ worst-case time solution to the compressed search problem with the two-dimensional LZ compression algorithm, where σ is $\min(|P|, |\Sigma|)$, and Σ is the alphabet. The amount of space used is $O(m)$, for an $m \times m$ size pattern. $O(m)$ is the best compression achievable for any $m \times m$ sized pattern under the two-dimensional LZ compression.*

The algorithm of [8] can be applied to any two-dimensional compressed text, in which the compression technique allows sequential decompression in small space.

Applications

The problem has many applications since two-dimensional data appears in many different types of compression. The two compressions discussed here are the run-length compression, used by fax transmissions, and the LZ compression, used by GIF.

Open Problems

Any lossless two-dimensional compression used, especially one with a large compression ratio, presents the problem of enabling the search without uncompressing the data for saving of both time and space.

Searching in two-dimensional lossy compressions will be a major challenge. Initial steps in this direction can be found in [15,14], where JPEG compression is considered.

Cross References

▶ Compressed Pattern Matching
▶ Multidimensional String Matching

Recommended Reading

1. Amir, A., Benson, G.: Efficient two dimensional compressed matching. In: Proceeding of Data Compression Conference, Snow Bird, Utah, 1992, pp. 279–288
2. Amir, A., Benson, G.: Two-dimensional periodicity and its application. Proceeding of 3rd Symposium on Discrete Algorithms, Orlando, FL, 1992, pp. 440–452
3. Amir, A., Benson, G.: Two-dimensional periodicity and its application. SIAM J. Comput. **27**(1), 90–106 (1998)

4. Amir, A., Benson, G., Farach, M.: The truth, the whole truth, and nothing but the truth: Alphabet independent two dimensional witness table construction.Technical Report GIT-CC-92/52, Georgia Institute of Technology (1992)
5. Amir, A., Benson, G., Farach, M.: An alphabet independent approach to two dimensional pattern matching. SIAM J. Comput. **23**(2), 313–323 (1994)
6. Amir, A., Benson, G., Farach, M.: Optimal two-dimensional compressed matching. J. Algorithms **24**(2), 354–379 (1997)
7. Amir, A., Benson, G., Farach, M.: Optimal parallel two dimensional text searching on a crew pram. Inf. Comput. **144**(1), 1–17 (1998)
8. Amir, A., Landau, G., Sokol, D.: Inplace 2d matching in compressed images. J. Algorithms **49**(2), 240–261 (2003)
9. Amir, A., Landau, G., Sokol, D.: Inplace run-length 2d compressed search. Theor. Comput. Sci. **290**(3), 1361–1383 (2003)
10. Berman, P., Karpinski, M., Larmore, L., Plandowski, W., Rytter, W.: On the complexity of pattern matching for highly compressed two dimensional texts. Proceeding of 8th Annual Symposium on Combinatorial Pattern Matching (CPM 97). LNCS, vol. 1264, pp. 40–51. Springer, Berlin (1997)
11. Crochemore, M., Galil, Z., Gasieniec, L., Hariharan, R., Muthukrishnan, S., Park, K., Ramesh, H., Rytter, W.: Parallel two-dimensional pattern matching. In: Proceeding of 34th Annual IEEE FOCS, 1993, pp. 248–258
12. Galil, Z., Park, K.: Alphabet-independent two-dimensional witness computation. SIAM. J. Comput. **25**(5), 907–935 (1996)
13. Hennessy, J.L., Patterson, D.A.: Computer Architeture: A Quantitative Approach, 2nd edn. Morgan Kaufmann, San Francisco, CA (1996)
14. Klein, S.T., Shapira, D.: Compressed pattern matching in jpeg images. In: Proceeding Prague Stringology conference, 2005, pp. 125–134
15. Klein, S.T., Wiseman, Y.: Parallel huffman decoding with applications to jpeg files. Comput. J. **46**(5), 487–497 (2003)
16. Park, K., Galil, Z.: Truly alphabet-independent two-dimensional pattern matching. In: Proceeding 33rd IEEE FOCS, 1992, pp. 247–256
17. Régnier, M., Rostami, L.: A unifying look at d-dimensional periodicities and space coverings. In: 4th Symp. on Combinatorial Pattern Matching, 15, 1993
18. Vishkin, U.: Optimal parallel pattern matching in strings. In: Proceeding 12th ICALP, 1985, pp. 91–113
19. Welch, T.A.: A technique for high-performance data compression. IEEE Comput. **17**, 8–19 (1984)
20. Ziv, J.: Personal communication (1995)

Multidimensional String Matching

1999; Kärkkäinen, Ukkonen

Juha Kärkkäinen, Esko Ukkonen
Department of Computer Science, University of Helsinki, Helsinki, Finland

Keywords and Synonyms

Multidimensional array matching; Image matching; Template registration

Problem Definition

Given two two-dimensional arrays, the *text* $T[1 \dots n, 1 \dots n]$ and the *pattern* $P[1 \dots m, 1 \dots m]$, $m \leq n$, both with element values from *alphabet* Σ of size σ, the basic *two-dimensional string matching* (2DSM) problem is to find all *occurrences* of P in T, i. e., all $m \times m$ subarrays of T that are identical to P. In addition to the basic problem, several types of generalizations are considered: *approximate matching* (allow local errors), *invariant matching* (allow global transformations), *indexed matching* (preprocess the text), and *multidimensional matching*.

In approximate matching, an occurrence is a subarray S of the text, whose *distance* $d(S, P)$ from the pattern does not exceed a threshold k. Different distance measures lead to different variants of the problem. When no distance is explicitly mentioned, the *Hamming distance*, the number of mismatching elements, is assumed.

For one-dimensional strings, the most common distance is the Levenshtein distance, the minimum number of insertions, deletions and substitutions for transforming one string into the other. A simple generalization to two dimensions is the *Krithivasan–Sitalakshmi (KS) distance*, which is the sum of row-wise Levenshtein distances. Baeza-Yates and Navarro [6] introduced several other generalizations, one of which, the *RC distance*, is defined as follows. A two-dimensional array can be decomposed into a sequence of rows and columns by removing either the last row or the last column from the array until nothing is left. Different decompositions are possible depending on whether a row or a column is removed at each step. The RC distance is the minimum cost of transforming a decomposition of one array into a decomposition of the other, where the minimum is taken over all possible decompositions as well as all possible transformations. A transformation consists of insertions, deletions and modifications of rows and columns. The cost of inserting or deleting a row/column is the length of the row/column, and the cost of modification is the Levenshtein distance between the original and the modified row/column.

The invariant matching problems search for occurrences that match the pattern after some global transformation of the pattern. In the *scaling invariant matching* problem, an occurrence is a subarray that matches the pattern scaled by some factor. If only integral scaling factors are allowed, the definition of the problem is obvious. For real-valued scaling, a refined model is needed, where the text and pattern elements, called *pixels* in this case, are unit squares on a plane. Scaling the pattern means stretching the pixels. An occurrence is a matching M between text pixels and pattern pixels. The scaled pattern is placed on top of the text with one corner aligned, and each text pixel $T[r, s]$, whose center is covered by the pattern, is matched with the covering pattern pixel $P[r', s']$, i. e., $([r, s], [r', s']) \in M$.

In the *rotation invariant matching* problem, too, an occurrence is a matching between text pixels and pattern pixels. This time the center of the pattern is placed at the center of a text pixel and the pattern is rotated around the center. The matching is again defined by which pattern pixels cover which text pixel centers.

In the *indexed* form of the problems, the text can be preprocessed to speed up the matching. The preprocessing and matching complexities are reported separately.

All the problems can be generalized to more than two dimensions. In the d-dimensional problem, the text is an n^d array and the pattern an m^d array. The focus is on two dimensions, but multidimensional generalizations of the results are mentioned when they exist.

Many other variants of the problems are omitted here due to lack of space. Some of them as well as some of the results in this entry are surveyed by Amir [1]. A wider range of problems as well as traditional image processing techniques for solving them can be found in [9].

Key Results

The classical solution to the 2DSM problem by Bird [8] and independently by Baker [7] reduces the problem to one-dimensional string matching. It has two phases:

1. Find all occurrences of pattern rows on the text rows and mark them. This takes $\mathcal{O}(n^2 \log \min(m, \sigma))$ time using the Aho-Corasick algorithm. On an integer alphabet $\Sigma = \{0, 1, \dots, \sigma - 1\}$, the time can be improved to $\mathcal{O}(n^2 + m^2 \min(m, \sigma) + \sigma)$ using $\mathcal{O}(m^2 \min(m, \sigma) + \sigma)$ space.
2. The pattern is considered a sequence of m rows and each $n \times m$ subarray of the text a sequence of n rows. The Knuth–Morris–Pratt string matching algorithm is used for finding the occurrences of the pattern in each subarray. The algorithm makes $\mathcal{O}(n)$ row comparisons for each of the $n - m + 1$ subarrays. With the markings from Step 1, a row comparison can be done in constant time, giving $\mathcal{O}(n^2)$ time complexity for Step 2.

The time complexity of the Bird–Baker algorithm is linear if the alphabet size σ is constant. The algorithm of Amir, Benson and Farach [2] (with improvements by Galil and Park [14]) achieves linear time independent of the alphabet size using a quite different kind of algorithm based on string matching by duels and two-dimensional periodicity.

Theorem 1 (Bird [8]; Baker [7]; Amir, Benson and Farach [2]) *The 2DSM problem can be solved in the optimal $\mathcal{O}(n^2)$ worst-case time.*

The Bird–Baker algorithm generalizes straightforwardly into higher dimensions by repeated application of Step 1 to reduce a problem in d dimensions into $n - m + 1$ problems in $d - 1$ dimensions. The time complexity is $\mathcal{O}(dn^d \log m^d)$. The Amir–Benson–Farach algorithm has been generalized to three dimensions with the time complexity $\mathcal{O}(n^3)$ [13].

The average-case complexity of the 2DSM problem was studied by Kärkkäinen and Ukkonen [15], who proved a lower bound and gave an algorithm matching the bound.

Theorem 2 (Kärkkäinen and Ukkonen [15]) *The 2DSM-problem can be solved in the optimal $\mathcal{O}(n^2(\log_\sigma m)/m^2)$ average-case time.*

The result (both lower and upper bound) generalizes to the d-dimensional case with the $\Theta(n^d \log_\sigma m/m^d)$ average-case time complexity.

Amir and Landau [5] give algorithms for approximate 2DSM problems for both the Hamming distance and the KS distance. The RC model was developed and studied by Baeza–Yates and Navarro [6].

Theorem 3 (Amir and Landau [5]; Baeza–Yates and Navarro [6]) *The approximate 2DSM problem can be solved in $\mathcal{O}(kn^2)$ worst-case time for the Hamming distance, in $\mathcal{O}(k^2 n^2)$ worst-case time for the KS distance, and in $\mathcal{O}(k^2 mn^2)$ worst-case time for the RC distance.*

The results for the KS and RC distances generalize to d dimensions with the time complexities $\mathcal{O}(k(k + d)n^d)$ and $\mathcal{O}(d! m^{2d} n^d)$, respectively.

Approximate matching algorithms with good average-case complexity are described by Kärkkäinen and Ukkonen [15] for the Hamming distance, and by Baeza-Yates and Navarro [6] for the KS and RC distances.

Theorem 4 (Kärkkäinen and Ukkonen [15]; Baeza–Yates and Navarro [6]) *The approximate 2DSM problem can be solved in $\mathcal{O}(kn^2(\log_\sigma m)/m^2)$ average-case time for the Hamming and KS distances, and in $\mathcal{O}(n^2/m)$ average-case time for the RC distance.*

The results for the Hamming and the RC distance have d-dimensional generalizations with the time complexities $\mathcal{O}(kn^d(\log_\sigma m^d)/m^d)$ and $\mathcal{O}(kn^d/m^{d-1})$, respectively.

The scaling and rotation invariant 2DSM problems involve a continuous valued parameter (scaling factor or rotation angle). However, the corresponding matching between text and pattern pixels changes only at certain points, and there are only $\mathcal{O}(nm)$ effectively distinct scales and $\mathcal{O}(m^3)$ effectively distinct rotation angles. A separate search for each distinct scale or rotation would give algorithms with time complexities $\mathcal{O}(n^3 m)$ and $\mathcal{O}(n^2 m^3)$, but faster algorithms exist.

Theorem 5 (Amir and Chencinski [3]; Amir, Kapah and Tsur [4]) *The scaling invariant 2DSM problem can be solved in $\mathcal{O}(n^2 m)$ worst-case time, and the rotation invariant 2DSM problem in $\mathcal{O}(n^2 m^2)$ worst-case time.*

Fast average-case algorithms for the rotation invariant problem are described by Fredriksson, Navarro and Ukkonen [11]. They also consider approximate matching versions.

Theorem 6 (Fredriksson, Navarro and Ukkonen [11]) *The rotation invariant 2DSM problem can be solved in the optimal $\mathcal{O}(n^2(\log_\sigma m)/m^2)$ average-case time. The rotation invariant approximate 2DSM problem can be solved in the optimal $\mathcal{O}(n^2(k + \log_\sigma m)/m^2)$ average-case time.*

Fredriksson, Navarro and Ukkonen [11] also consider rotation invariant matching in d dimensions.

Indexed matching is based on two-dimensional suffix trees and arrays, which are the subject of another entry *2D-Pattern Indexing*. Their properties are similar to one-dimensional suffix trees and arrays.

Theorem 7 (Kim and Park [16]) *The text can be preprocessed in $\mathcal{O}(n^2)$ time so that subsequently a 2DSM query can be answered in $\mathcal{O}(m^2 \log \sigma)$ time or in $\mathcal{O}(m^2 + \log n)$ time.*

Fredriksson, Navarro and Ukkonen [11] describe an index suitable for rotation invariant matching.

Theorem 8 (Fredriksson, Navarro and Ukkonen [11]) *The text can be preprocessed in $\mathcal{O}(n^2)$ time so that subsequently a rotation invariant 2DSM query can be answered in $\mathcal{O}((\log_\sigma n)^{5/2})$ average-case time and a rotation invariant approximate 2DSM query can be answered in $\mathcal{O}((2\log_\sigma n)^{k+3/2}\sigma^k)$ average-case time.*

Applications

The main application area is pattern matching in images, particularly applications where the point of view in the image is well-defined, such as aerial and astronomical photography, optical character recognition, and biomedical imaging. Even three-dimensional problems arise in biomedical applications [12].

Open Problems

Many combinations of the different variants of the problem have not been studied. Combining scaling and rotation invariance is an example. With rotation invariant approximate matching under the RC distance even the problem needs further specification.

Experimental Results

No conclusive results exist though some experiments are reported in [10,12,15].

Cross References

Many of the problems and their solutions are related to one-dimensional string matching problems and techniques described in the entry ▶ Sequential Approximate String Matching. The construction of text index structures is described in ▶ Two-Dimensional Pattern Indexing. Matching on a compressed text without decompression is considered in ▶ Multidimensional Compressed Pattern Matching.

Recommended Reading

1. Amir, A.: Theoretical issues of searching aerial photographs: a bird's eye view. Int. J. Found. Comput. Sci. **16**, 1075–1097 (2005)
2. Amir, A., Benson, G., Farach, M.: An alphabet independent approach to two-dimensional pattern matching. SIAM J. Comput. **23**, 313–323 (1994)
3. Amir, A., Chencinski, E.: Faster two dimensional scaled matching. In: Proc. 17th Annual Symposium on Combinatorial Pattern Matching. LNCS, vol. 4009, pp. 200–210. Springer, Berlin (2006)
4. Amir, A., Kapah, O., Tsur, D.: Faster two dimensional pattern matching with rotations. In: Proc. 15th Annual Symposium on Combinatorial Pattern Matching. LNCS, vol. 3109, pp. 409–419. Springer, Berlin (2004)
5. Amir, A., Landau, G.M.: Fast parallel and serial multidimensional approximate array matching. Theoretical Comput. Sci. **81**, 97–115 (1991)
6. Baeza-Yates, R., Navarro, G.: New models and algorithms for multidimensional approximate pattern matching. J. Discret. Algorithms **1**, 21–49 (2000)
7. Baker, T.P.: A technique for extending rapid exact-match string matching to arrays of more than one dimension. SIAM J. Comput. **7**, 533–541 (1978)
8. Bird, R.S.: Two dimensional pattern matching. Inf. Process. Lett. **6**, 168–170 (1977)
9. Brown, L.G.: A survey of image registration techniques. ACM Computing Surveys **24**, 325–376 (1992)
10. Fredriksson, K., Navarro, G., Ukkonen, E.: Faster than FFT: Rotation invariant combinatorial template matching. In: Pandalai, S. (ed.) Recent Research Developments in Pattern Recognition, vol. II, pp. 75–112. Transworld Research Network, Trivandrum, India (2002)
11. Fredriksson, K., Navarro, G., Ukkonen, E.: Sequential and indexed two-dimensional combinatorial template matching allowing rotations. Theoretical Comput. Sci. **347**, 239–275 (2005)

12. Fredriksson, K., Ukkonen, E.: Combinatorial methods for approximate pattern matching under rotations and translations in 3D arrays. In: Proc. 7th International Symposium on String Processing and Information Retrieval, pp. 96–104. IEEE Computer Society, Washington, DC (2000)
13. Galil, Z., Park, J.G., Park, K.: Three-dimensional periodicity and its application to pattern matching. SIAM J. Discret. Math. **18**, 362–381 (2004)
14. Galil, Z., Park, K.: Alphabet-independent two-dimensional witness computation. SIAM J. Comput. **25**, 907–935 (1996)
15. Kärkkäinen, J., Ukkonen, E.: Two- and higher-dimensional pattern matching in optimal expected time. SIAM J. Comput. **29**, 571–589 (1999)
16. Kim, D.K., Park, K.: Linear-time construction of two-dimensional suffix trees. In: Proc. 26th International Colloquium on Automata Languages and Programming. LNCS, vol. 1644, pp. 463–472. Springer, Berlin (1999)

Multi-Hop Radio Networks, Ad Hoc Networks

► Randomized Broadcasting in Radio Networks

Multi-level Feedback Queues

1968; Coffman, Kleinrock

NIKHIL BANSAL
IBM Research, IBM, Yorktown Heights, NY, USA

Keywords and Synonyms

Fairness; Low sojourn times; Scheduling with unknown job sizes

Problem Definition

The problem is concerned with scheduling dynamically arriving jobs in the scenario when the processing requirements of jobs are unknown to the scheduler. This is a classic problem that arises for example in CPU scheduling, where users submit jobs (various commands to the operating system) over time. The scheduler is only aware of the existence of the job and does not know how long it will take to execute, and the goal is to schedule jobs to provide good quality of service to the users. Formally, this note considers the average flow time measure, defined as the average duration of time since a job is released until its processing requirement is met.

Notations

Let $\mathcal{J} = \{1, 2, \dots, n\}$ denote the set of jobs in the input instance. Each job j is characterized by its release time r_j and its processing requirement p_j. In the online setting, job j is revealed to the scheduler only at time r_j. A further restriction is the *non-clairvoyant* setting, where only the existence of job j is revealed at r_j, in particular the scheduler does not know p_j until the job meets its processing requirement and leaves the system. Given a schedule, the completion time c_j of a job is the earliest time at which job j receives p_j amount of service. The flow time f_j of j is defined as $c_j - r_j$. A schedule is said to be preemptive, if a job can be interrupted arbitrarily, and its execution can be resumed later from the point of interruption without any penalty. It is well known that preemption is necessary to obtain reasonable guarantees even in the offline setting [4].

There are several natural non-clairvoyant algorithms such as First Come First Served, Processor Sharing (work on all current unfinished jobs at equal rate), Shortest Elapsed Time First (work on job that has received least amount of service thus far). Coffman and Kleinrock [2] proposed another natural algorithm known as the Multi-Level Feedback Queueing (MLF). MLF works as follows: There are queues $Q_0, Q_1, Q_2, \dots$ and thresholds $0 < t_0 < t_1 < t_2 \dots$. Initially upon arrival, a job is placed in Q_0. When a job in Q_i receives t_i amount of cumulative service, it is moved to Q_{i+1}. The algorithm at any time works on the lowest numbered non-empty queue. Coffman and Kleinrock analyzed MLF in a queuing theoretic setting, where the jobs arrive according to a Poisson process and the processing requirements are chosen identically and independently from a known probability distribution.

Recall that the online Shortest Remaining Processing Time (SRPT) algorithm, that at any time works on the job with the least remaining processing time, produces an optimum schedule. However, SRPT requires the knowledge of job sizes and hence is not non-clairvoyant. Since a non-clairvoyant algorithm only knows a lower bound on a jobs size (determined by the amount of service it has received thus far), MLF tries to mimic SRPT by favoring jobs that have received the least service thus far.

Key Results

While non-clairvoyant algorithms have been studied extensively in the queuing theoretic setting for many decades, this notion was considered relatively recently in the context of competitive analysis by Motwani, Phillips and Torng [5]. As in traditional competitive analysis, a non-clairvoyant algorithm is called c-competitive if for every input instance, its performance is no worse than c times than optimum offline solution for that instance. Motwani, Phillips and Torng showed the following.

Theorem 1 ([5]) *For the problem of minimizing average flow time on a single machine, any deterministic nonclairvoyant algorithm must have a competitive ratio of at least $\Omega(n^{1/3})$ and any randomized algorithm must have a competitive ratio of at least $\Omega(\log n)$, where n is number of jobs in the instance.*

It is not too surprising that any deterministic algorithm must have a poor competitive ratio. For example, consider MLF where the thresholds are powers of 2, i. e. $1, 2, 4, \ldots$. Say $n = 2^k$ jobs of size $2^k + 1$ each arrive at times $0, 2^k, 2 \cdot 2^k, \ldots, (2^k - 1)2^k$ respectively. Then, it is easily verified that the average flow time under MLF is $\Omega(n^2)$, where as the average flow time is under the optimum algorithm is $\Omega(n)$.

Note that MLF performs poorly on the above instance since all jobs are stuck till the end with just one unit of work remaining. Interestingly, Kalyanasundaram and Pruhs [3] designed a randomized variant of MLF (known as RMLF) and proved that its competitive ratio is almost optimum. For each job j, and for each queue Q_i, the RMLF algorithm sets a threshold $t_{i,j}$ randomly and independently according to a truncated exponential distribution. Roughly speaking, setting a random threshold ensures that if a job is stuck in a queue, then its remaining processing is a reasonable fraction of its original processing time.

Theorem 2 ([3]) *The RMLF algorithm is $O(\log n \log\log n)$ competitive against an oblivious adversary. Moreover, the RMLF algorithm is $O(\log n \log\log n)$ competitive even against an adaptive adversary provided the adversary chooses all the job sizes in advance.*

Later, Becchetti and Leonardi [1] showed that in fact the RMLF is optimally competitive up to constant factors. They also analyzed RMLF on identical parallel machines.

Theorem 3 ([1]) *The RMLF algorithm is $O(\log n)$ competitive for a single machine. For multiple identical machines, RMLF achieves a competitive ratio of $O(\log n \log(\frac{n}{m}))$, where m is the number of machines.*

Applications

MLF and its variants are widely used in operating systems [6,7]. These algorithms are not only close to optimum with respect to flow time, but also have other attractive properties such as the amortized number of preemptions is logarithmic (preemptions occur only if a job arrives or departs or moves to another queue).

Open Problems

It is not known whether there exists a $o(n)$-competitive deterministic algorithm. It would be interesting to close the gap between the upper and lower bounds for this case. Often in real systems, even though the scheduler may not know the exact job size, it might have some information about its distribution based on historical data. An interesting direction of research could be to design and analyze algorithms that use this information.

Cross References

► Flow Time Minimization
► Minimum Flow Time
► Shortest Elapsed Time First Scheduling

Recommended Reading

1. Becchetti, L., Leonardi, S.: Nonclairvoyant scheduling to minimize the total flow time on single and parallel machines. J. ACM (JACM) **51**(4), 517–539 (2004)
2. Coffman, E.G., Kleinrock, L.: Feedback Queueing Models for Time-Shared Systems. J. ACM (JACM) **15**(4), 549–576 (1968)
3. Kalyanasundaram, B., Pruhs, K.: Minimizing flow time nonclairvoyantly. J. ACM (JACM) **50**(4), 551–567 (2003)
4. Kellerer, H., Tautenhahn, T., Woeginger, G.J.: Approximability and Nonapproximability Results for Minimizing Total Flow Time on a Single Machine. SIAM J. Comput. **28**(4), 1155–1166 (1999)
5. Motwani, R., Phillips, S., Torng, E.: Non-Clairvoyant Scheduling. Theor. Comput. Sci. **130**(1), 17–47 (1994)
6. Nutt, G.: Operating System Projects Using Windows NT. Addison Wesley, Reading (1999)
7. Tanenbaum, A.S.: Modern Operating Systems. Prentice-Hall Inc., Upper Saddle River (1992)

Multiple String Alignment

► Efficient Methods for Multiple Sequence Alignment with Guaranteed Error Bounds

Multiple Unit Auctions with Budget Constraint

2005; Borgs, Chayes, Immorlica, Mahdian, Saberi
2006; Abrams

TIAN-MING BU
Department of Computer Science, Fudan University, Shanghai, China

Problem Definition

In this problem, an auctioneer would like to sell an idiosyncratic commodity with m copies to n bidders, denoted by $i = 1, 2, \ldots, n$. Each bidder i has two kinds of privately known information: $t_i^u \in \mathbb{R}^+$, $t_i^b \in \mathbb{R}^+$. $t_i^u \in \mathbb{R}^+$ represents the price buyer i is willing to pay per copy of the commodity and $t_i^b \in \mathbb{R}^+$ represents i's budget.

Then a *one-round sealed-bid* auction proceeds as follows. *Simultaneously* all the bidders submit their bids to the auctioneer. When receiving the reported unit value vector $\mathbf{u} = (u_1, \ldots, u_n)$ and the reported budget vector $\mathbf{b} = (b_1, \ldots, b_n)$ of bids, the auctioneer computes and outputs the allocation vector $\mathbf{x} = (x_1, \ldots, x_n)$ and the price vector $\mathbf{p} = (p_1, \ldots, p_n)$. Each element of the allocation vector indicates the number of copies allocated to the corresponding bidder. If bidder i receives x_i copies of the commodity, he pays the auctioneer $p_i x_i$. Then bidder i's total payoff is $(t_i^u - p_i)x_i$ if $x_i p_i \le t_i^b$ and $-\infty$ otherwise. Correspondingly, the revenue of the auctioneer is $\mathcal{A}(\mathbf{u}, \mathbf{b}, m) = \sum_i p_i x_i$.

If each bidder submits his privately *true* unit value t_i^u and budget t_i^b to the auctioneer, the auctioneer can determine the single price $p_{\mathcal{F}}$ (i. e., $\forall i, p_i = p_{\mathcal{F}}$) and the allocation vector which maximize the auctioneer's revenue. This optimal single price revenue is denoted by $\mathcal{F}(\mathbf{u}, \mathbf{b}, m)$.

Interestingly, in this problem, we assume bidders have free will, and have complete knowledge of the auction mechanism. Bidders would just report the bid (maybe different from his corresponding privately true values) which could maximize his payoff according to the auction mechanism.

So the objective of the problem is to design a *truthful* auction satisfying *voluntary participation* to raise the auctioneer's revenue as much as possible. An auction is *truthful* if for every bidder i, bidding his true valuation would maximize his payoff, regardless of the bids submitted by the other bidders [8,9]. An auction satisfies *voluntary participation* if each bidder's payoff is guaranteed to be non-negative if he reports his bid truthfully. The performance of the auction $\mathcal{A}$ is determined by competitive ratio β which is defined as the upper bound of $\frac{\mathcal{F}(\mathbf{u},\mathbf{b},m)}{\mathcal{A}(\mathbf{u},\mathbf{b},m)}$ [5]. Clearly, the smaller the competitive ratio β is, the better the auction $\mathcal{A}$ is.

Definition (Multiple Unit Auctions with Budget Constraint)

INPUT: the number of copies m, the submitted unit value vector $\mathbf{u}$, the submitted budget vector $\mathbf{b}$.

OUTPUT: the allocation vector $\mathbf{x}$ and the price vector $\mathbf{p}$.

CONSTRAINTS:

(a) Truthful

(b) Voluntary participation

(c) $\sum_i x_i \le m$.

Key Results

Let $b_{\max}$ denote the largest budget amongst the bidders receiving copies in the optimal solution and define $\alpha = \frac{\mathcal{F}}{b_{\max}}$.

Theorem 1 ([2]) *A truthful auction satisfying voluntary participation with competitive ratio* $1/\max_{0<\delta<1}\{(1-\delta)(1-2\,\mathrm{e}^{-\frac{\alpha\delta^2}{36}})\}$ *can be designed.*

Theorem 2 ([1]) *A truthful auction satisfying voluntary participation with competitive ratio* $\frac{4\alpha}{\alpha-1}$ *can be designed.*

Theorem 3 ([1]) *If* α *is known in advance, then a truthful auction satisfying voluntary participation with competitive ratio* $\frac{(x\alpha+1)\alpha}{(x\alpha-1)^2}$ *can be designed, where* $x = \frac{\alpha-1+((\alpha-1)^2-4\alpha)^{1/2}}{2\alpha}$.

Theorem 4 ([1]) *For any truthful randomized auction* $\mathcal{A}$ *satisfying voluntary participation, the competitive ratio is at least* $2-\epsilon$ *when* $\alpha \ge 2$.

Applications

This problem is motivated by the development of the IT industry and the popularization of auctions, especially, auctions on the Internet. The multiple copy auction of relatively low-value goods, such as the auction of online ads for search terms to bidders with budget constraint, is assuming a very important role. Companies such as Google and Yahoo!'s revenue depends almost on certain types of auctions.

All previous work concerning auctions with budget constraint only focused on the traditional physical world where the object to sell is almost unique, such as antiques, paintings, land and nature resources. More specifically, [4] studied the problem of single unit, single bidder with a budget. [6] studied the model of single unit, multiple bidders with common public budget. [7] studied the problem of single unit, multiple bidders with flexible budgets.

Recently, [3] extended this problem so that the auctioneer could sell unlimited copies of goods and the bidders have both budget and copy constraints. This general model is especially suitable for digital goods, which can produce unlimited copies with marginal cost zero, such as license sales, mp3 copies, online advertisements, etc. Further, the auction mechanism designed in [3] could obtain a similar competitive ratio.

Cross References

► Competitive Auction

Recommended Reading

1. Abrams, Z.: Revenue maximization when bidders have budgets. In: Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA-06), Miami, Florida, 22–26 Jan 2006, pp. 1074–1082. ACM Press, New York
2. Borgs, C., Chayes, J.T., Immorlica, N., Mahdian, M., Saberi, A.: Multi-unit auctions with budget-constrained bidders. In: ACM Conference on Electronic Commerce (EC-05), 2005, pp. 44–51
3. Bu, T.-M., Qi, Q., Sun, A.W.: Unconditional competitive auctions with copy and budget constraints. In: Spirakis, P.G., Mavronicolas, M., Kontogiannis, S.C. (eds.) Internet and Network Economics, 2nd International Workshop, WINE 2006, Patras, Greece, 15–17 Dec 2006. Lecture Notes in Computer Science, vol. 4286, pp. 16–26. Springer, Berlin (2006)
4. Che, Y.-K., Gale, I.: Standard auctions with financially constrained bidders. Rev. Econ. Stud. **65**(1), 1–21 (1998)
5. Goldberg, A.V., Hartline, J.D., Karlin, A.R., Wright, A.: Competitive auctions. Games Econ. Behav. **55**(2), 242–269 (2006)
6. Laffont, J.-J., Robert, J.: Optimal auction with financially constrained buyers. Econ. Lett. **52**, 181–186 (1996)
7. Maskin, E.S.: Auctions, development, and privatization: Efficient auctions with liquidity-constrained buyers. Eur. Econ. Rev. **44**(4–6), 667–681 (2000)
8. Nisan, N. Ronen, A.: Algorithmic mechanism design. In: Proceedings of the 31st Annual ACM Symposium on Theory of Computing (STOC-99), pp. 129–140. Association for Computing Machinery, New York (1999)
9. Parkes, D.C.: Chapter 2: Iterative Combinatorial Auctions. Ph. D. thesis, University of Pennsylvania (2004)

Multiplex PCR for Gap Closing (Whole-genome Assembly)

2002; Alon, Beigel, Kasif, Rudich, Sudakov

VERA ASODI
Center for the Mathematics of Information, California Institute of Technology, Pasadena, CA, USA

Keywords and Synonyms

Whole genome assemble; Multiplex PCR

Problem Definition

This problem is motivated by an important and timely application in computational biology that arises in whole-genome shotgun sequencing. Shotgun sequencing is a high throughput technique that has resulted in the sequencing of a large number of bacterial genomes as well as Drosophila (fruit fly) and Mouse and the celebrated Human genome (at Celera) (see, e. g. [8]). In all such projects, one is left with a collection of DNA fragments. These fragments are subsequently assembled, in-silico, by a computational algorithm. The typical assembly algorithm repeatedly merges overlapping fragments into longer fragments called contigs. For various biological and computational reasons some regions of the DNA cannot covered by the contigs. Thus, the contigs must be ordered and oriented and the gaps between them must be sequenced using slower, more tedious methods. For further details see, e. g., [3]. When the number of gaps is small (e. g., less than ten) biologists often use combinatorial PCR. This technique initiates a set of "bi-directional molecular walks" along the gaps in the sequence; these walks are facilitated by PCR. In order to initiate the molecular walks biologists use primers. Primers are designed so that they bind to unique (with respect to the entire DNA sequence) templates occurring at the end of each contig. A primer (at the right temperature and concentration) anneals to the designated unique DNA substring and promotes copying of the template starting from the primer binding site, initiating a one-directional walk along the gap in the DNA sequence. A PCR reaction occurs, and can be observed as a DNA ladder, when two primers that bind to positions on two ends of the same gap are placed in the same test tube.

If there are N contigs, the combinatorial (exhaustive) PCR technique tests all possible pairs (quadratically many) of $2N$ primers by placing two primers per tube with the original uncut DNA strand. PCR products can be detected using gels or they can be read using sequencing technology or DNA mass-spectometry. When the number of gaps is large, the quadratic number of PCR experiments is prohibitive, so primers are pooled using $K > 2$ primers per tube; this technique is called multiplex PCR [4]. This problem deals with finding optimal strategies for pooling the primers to minimize the number of biological experiments needed in the gap-closing process.

This problem can be modeled as the problem of identifying or learning a hidden matching given a vertex set V and an allowed query operation: for a subset $F \subseteq V$, the query Q_F is "does F contain at least one edge of the matching"? In this formulation each vertex represents a primer, an edge of the matching represents a reaction, and the query represents checking for a reaction when a set of primers are combined in a test tube. The objective is to identify the matching asking as few queries as possible, that is performing as few tests as possible. For further discussion of this model see [3,7].

This problem is of interest even in the deterministic, fully non-adaptive case. A family $\mathcal{F}$ of subsets of a vertex set V solves the matching problem on V if for any two distinct matchings M_1 and M_2 on V there is at least one $F \in \mathcal{F}$ that contains an edge of one of the matchings and does not contain any edge of the other. Obviously, any such family enables learning an unknown matching deterministically and non-adaptively, by asking the questions Q_F for each $F \in \mathcal{F}$. The objective here is to determine the minimum possible cardinality of a family that solves the matching problem on a set of n vertices.

Other interesting variants of this problem are when the algorithm may be randomized, or when it is adaptive, that is when the queries are asked in k rounds, and the queries of each round may depend on the answers from the previous rounds.

Key Results

In [2], the authors study the number of queries needed to learn a hidden matching in several models. Following is a summary of the main results presented in this paper.

The trivial upper bound on the size of a family that solves the matching problem on n vertices is $\binom{n}{2}$, achieved by the family of all pairs of vertices. Theorem 1 shows that in the deterministic non-adaptive setting one cannot do much better than this, namely, that the trivial upper bound is tight up to a constant factor. Theorem 2 improves this upper bound by showing a family of approximately half that size that solves the matching problem.

Theorem 1 *For every $n > 2$, every family $\mathcal{F}$ that solves the matching problem on n vertices satisfies*

$$|\mathcal{F}| \geq \frac{49}{153}\binom{n}{2} .$$

Theorem 2 *For every n there exists a family of size*

$$\left(\frac{1}{2} + o(1)\right)\binom{n}{2}$$

that solves the matching problem on n vertices.

Theorem 3 shows that one can do much better using randomized algorithms. That is, one can learn a hidden matching asking only $O(n \log n)$ queries, rather than order of n^2. These randomized algorithms make no errors, however, they might ask more queries with some small probability.

Theorem 3 *The matching problem on n vertices can be solved by probabilistic algorithms with the following parameters:*

- *2 rounds and $(1/(2\ln 2))n \log n(1 + o(1)) \approx 0.72n \log n$ queries*
- *1 round and $(1/\ln 2)n \log n(1 + o(1)) \approx 1.44n \log n$ queries.*

Finally, Theorem 4 considers adaptive algorithms. In this case there is a tradeoff between the number of queries and the number of rounds. The more rounds one allows, the fewer tests are needed, however, as each round can start only after the previous one is completed, this increases the running time of the entire procedure.

Theorem 4 *For all $3 \leq k \leq \log n$, there is a deterministic k-round algorithm for the matching problem on n vertices that asks*

$$O\left(n^{1+\frac{1}{2(k-1)}}(\log n)^{1+\frac{1}{k-1}}\right)$$

queries per round.

Applications

As described in Sect. "Problem Definition", this problem was motivated by the application of gap closing in whole-genome sequencing, where the vertices correspond to primers, the edges to PCR reactions between pairs of primers that bind to the two ends of a gap, and the queries to tests in which a set of primers are combined in a test tube.

This gap-closing problem can be stated more generally as follows. Given a set of chemicals, a guarantee that each chemical reacts with at most one of the others, and an experimental mechanism to determine whether a reaction occurs when several chemicals are combined in a test tube, the objective is to determine which pairs of chemicals react with each other with a minimum number of experiments.

Another generalization which may have more applications in molecular biology is when the hidden subgraph is not a matching but some other fixed graph, or a family of graphs. The paper [2], as well as some other related works (e. g. [1,5,6]), consider this generalization for other graphs. Some of these generalizations have other specific applications in molecular biology.

Open Problems

- Determine the smallest possible constant c such that there is a deterministic non-adaptive algorithm for the matching problem on n vertices that performs $c\binom{n}{2}(1 + o(1))$ queries.
- Find more efficient deterministic k-round algorithms or prove lower bounds for the number of queries in such algorithms.
- Find efficient algorithms and prove lower bounds for the generalization of the problem to graphs other than matchings.

Recommended Reading

1. Alon, N., Asodi, V.: Learning a hidden subgraph, ICALP. LNCS **3142**, 110–121 (2004). Also: SIAM J. Discret. Math. **18**, 697–712 (2005)
2. Alon, N., Beigel, R., Kasif, S., Rudich, S., Sudakov, B.: Learning a Hidden Matching, Proceedings of the 43rd IEEE FOCS, 2002, 197–206. Also: SIAM J. Computing **33**, 487–501 (2004)
3. Beigel, R., Alon, N., Apaydin, M.S., Fortnow, L., Kasif, S.: An optimal procedure for gap closing in whole genome shotgun sequencing. Proc. RECOMB, ACM Press pp. 22–30. (2001)
4. Burgart, L.J., Robinson, R.A., Heller, M.J., Wilke, W.W., Iakoubova, O.K., Cheville, J.C.: Multiplex polymerase chain reaction. Mod. Pathol. **5**, 320–323 (1992)
5. Grebinski, V., Kucherov, G.: Optimal Query Bounds for Reconstructing a Hamiltonian Cycle in Complete Graphs. Proc. 5th Israeli Symposium on Theoretical Computer Science, pp. 166–173. (1997)
6. Grebinski, V., Kucherov, G.: Reconstructing a Hamiltonian Cycle by Querying the Graph: Application to DNA Physical Mapping. Discret. Appl. Math. **88**, 147–165 (1998)
7. Tettelin, H., Radune, D., Kasif, S., Khouri, H., Salzberg, S.: Pipette Optimal Multiplexed PCR: Efficiently Closing Whole Genome Shotgun Sequencing Project. Genomics **62**, 500–507 (1999)
8. Venter, J.C., Adams, M.D., Sutton, G.G., Kerlavage, A.R., Smith, H.O., Hunkapiller, M.: Shotgun sequencing of the human genome. Science **280**, 1540–1542 (1998)

Multiway Cut

1998; Calinescu, Karloff, Rabani

GRUIA CALINESCU
Department of Computer Science, Illinois Institute of Technology, Chicago, IL, USA

Keywords and Synonyms

Multiterminal cut

Problem Definition

Given an undirected graph with edge costs and a subset of k nodes called *terminals*, a *multiway cut* is a subset of edges whose removal disconnects each terminal from the rest. MULTIWAY CUT is the problem of finding a multiway cut of minimum cost.

Previous Work

Dahlhaus, Johnson, Papadimitriou, Seymour, and Yannakakis [6] initiated the study of MULTIWAY CUT and proved that MULTIWAY CUT is MAX SNP-hard even when restricted to instances with three terminals and unit edge costs. Therefore, unless $P = NP$, there is no polynomial-time approximation scheme for MULTIWAY CUT. For $k = 2$, the problem is identical to the undirected version of the extensively studied s-t min-cut problem of Ford and Fulkerson, and thus has polynomial-time algorithms (see, e. g., [1]). Prior to this paper, the best (and essentially the only) approximation algorithm for $k \geq 3$ was due to the above-mentioned paper of Dahlhaus et al. They give a very simple combinatorial *isolation heuristic* that achieves an approximation ratio of $2(1 - 1/k)$. Specifically, for each terminal i, find a minimum-cost cut separating i from the remaining terminals, and then output the union of the $k - 1$ cheapest of the k cuts. For $k = 4$ and for $k = 8$, Alon (see [6]) observed that the isolation heuristic can be modified to give improved ratios of 4/3 and 12/7, respectively.

In special cases, far better results are known. For fixed k in planar graphs, the problem is solvable in polynomial time [6]. For trees and 2-trees, there are linear-time algorithms [5]. For dense unweighted graphs, there is a polynomial-time approximation scheme [2,8].

Key Results

Theorem 1 ([3]) *There is a deterministic polynomial time algorithm that finds a multiway cut of cost at most $(1.5 - 1/k)$ times the optimum multiway cut.*

The approximation algorithm from Theorem 1 is based on a novel linear programming relaxation described later. On the basis of the same linear program, the approximation ratio was subsequently improved to 1.3438 by Karger, Klein, Stein, Thorup, and Young [10]. For three terminals, [10] and Cheung, Cunningham, and Tang [4] give very different 12/11-approximation algorithms.

Two variations of the problem have been considered in the literature: Garg, Vazirani, and Yannakakis [9] obtain a $(2 - 2/k)$-approximation ratio for the node-weighted version, and Naor and Zosin [11] obtain 2-approximation for the case of directed graphs. It is known that any approximation ratio for these variations translates immediately into the same approximation ratio for VERTEX COVER, and thus it is hard to get any significant improvement over the approximation ratio of 2.

The algorithm from Theorem 1 appears next, giving a flavor of how this result is obtained. The complete proof of the approximation ratio is not long and appears in [3] or the book [12].

Notation

Let $G = (V, E)$ be an undirected graph on $V = \{1, 2, \ldots, n\}$ in which each edge $uv \in E$ has a non-negative cost $c(u, v) = c(v, u)$, and let $T = \{1, 2, \ldots, k\} \subseteq V$ be a set of *terminals*. MULTIWAY CUT is the problem of finding a minimum cost set $C \subseteq E$ such that in $(V, E \setminus C)$, each of the terminals $1, 2, \ldots, k$ is in a different component. Let $\mathrm{MWC} = \mathrm{MWC}(G)$ be the value of the optimal solution to MULTIWAY CUT.

Δ_k denotes the $(k-1)$-simplex, i. e., the $(k-1)$-dimensional convex polytope in $\mathbb{R}^k$ given by $\{x \in \mathbb{R}^k | (x \geq 0) \wedge (\sum_i x_i = 1)\}$.

For $x \in \mathbb{R}^k$, $\|x\|$ is its L_1 norm: $\|x\| = \sum_i |x_i|$. For $j = 1, 2, \ldots, k$, $e^j \in \mathbb{R}^k$ denotes the unit vector given by $(e^j)_j = 1$ and $(e^j)_i = 0$ for all $i \neq j$.

LP-Relaxation

The *simplex* relaxation for MULTIWAY CUT with edge costs has as variables k-dimensional real vectors x^u, defined for each vertex $u \in V$:

$$\text{Minimize } \frac{1}{2} \sum_{uv \in E} c(u,v) \cdot \|x^u - x^v\|$$

Subject to:

$$x^u \in \Delta_k \qquad \forall u \in V$$
$$x^t = e^t \qquad \forall t \in T.$$

In other words, the terminals stay at the vertices of the $(k-1)$-simplex, and the other nodes anywhere in the simplex, and measure an edge's length by the total variation distance between its endpoints. Clearly, placing all nodes at simplex vertices gives an integral solution: the lengths of edges are either 0 (if both endpoints are at the same vertex) or 1 (if the endpoints are at different vertices), and the removal of all unit length edges disconnects the graph into at least k components, each containing at most one terminal.

To solve this relaxation as a linear program, new variables are introduced: y^{uv}, defined for all $uv \in E$, and x_i^u, defined for all $u \in V$ and $i \in T$. Also new variables are y_i^{uv}, defined for all $i \in T$ and $uv \in E$. Then one writes the linear program:

$$\text{Minimize } \frac{1}{2} \sum_{uv \in E} c(u,v) y^{uv}$$

Subject to:

$$x^u \in \Delta_k \qquad \forall u \in V$$
$$x^t = e^t \qquad \forall t \in T$$
$$y^{uv} = \sum_{i \in T} y_i^{uv} \qquad \forall uv \in E$$
$$y_i^{uv} \geq x_i^u - x_i^v \qquad \forall uv \in E\,, \quad i \in T$$
$$y_i^{uv} \geq x_i^v - x_i^u \qquad \forall uv \in E\,, \quad i \in T\,.$$

It is easy to see that this linear program optimally solves the simplex relaxation above, by noticing that an optimal solution to the linear program can be assumed to put $y_i^{uv} = |x_i^u - x_i^v|$ and $y^{uv} = \|x^u - x^v\|$. Thus, solving the simplex relaxation can be done in polynomial time. This is the first step of the approximation algorithm. Clearly, the value Z^* of this solution is a lower bound on the cost of the minimum multiway cut MWC.

The second step of the algorithm is a rounding procedure which transforms a feasible solution of the simplex relaxation into an integral feasible solution. The rounding procedure below differs slightly from the one given in [3], but can be proven to give exactly the same solution. This variant is easier to present, although if one wants to prove the approximation ratio then the only way we know of is by showing that indeed this variant gives the same solution as the more complicated algorithm given in [3].

Rounding

Set $B(i, \rho) = \{u \in V \mid x_i^u > 1 - \rho\}$, the set of nodes suitably "close" to terminal i in the simplex. Choose a permutation $\sigma = \langle \sigma_1, \sigma_2, \ldots, \sigma_k \rangle$ to be either $\langle 1, 2, 3, \ldots, k-1, k \rangle$ or $\langle k-1, k-2, k-3, \ldots, 1, k \rangle$ with probability 1/2 each. Independently, choose $\rho \in (0, 1)$ uniformly at random. Then, process the terminals in the order $\sigma(1), \sigma(2), \sigma(3), \ldots, \sigma(k)$. For each j from 1 to $k-1$, place the nodes that remain in $B(\sigma_j, \rho)$ at e^{σ_j}. Place whatever nodes remain at the end at e^k. The following code specifies the rounding procedure more formally. $\bar{x}$ denotes the rounded (integral) solution.

```
1:  Let $\sigma = \langle 1, \ldots, k-3, k-2, k-1, k \rangle$ or $\langle k-1, k-2, k-3, \ldots, 1, k \rangle$, each with prob. 1/2
2:  Let $\rho$ be a random real in (0, 1) /* See the paragraph below. */
3:  for $j = 1$ to $k-1$ do
4:    for all $u$ such that $x^u \in B(\sigma_j, \rho) \setminus \cup_{i:i<j} B(\sigma_i, \rho)$ do
5:      $\bar{x}^u := e^{\sigma_j}$ /* assign node $u$ to terminal $\sigma_j$ */
6:    end for
7:  end for
8:  for all $u$ such that $x^u \notin \cup_{i:i<k} B(\sigma_i, \rho)$ do
9:    $\bar{x}^u := e^k$
10: end for
```

Multiway Cut, Algorithm 1
The Rounding Procedure

To derandomize and implement this algorithm in polynomial time, one tries both permutations σ and at most

$k(n+1)$ values of ρ. Indeed, for any permutation σ, two different values of ρ, $\rho_1 < \rho_2$, produce combinatorially distinct solutions only if there is a terminal i and a node u such that $x_i^u \in (1-\rho_2, 1-\rho_1]$. Thus, there are at most $k(n+1)$ "interesting" values of ρ, which can be determined easily by sorting the nodes according to each coordinate separately. The resulting discrete sample space for (σ, ρ) has size at most $2k(n+1)$, so one can search it exhaustively.

The analysis of the algorithm, however, is based on the randomized algorithm above, as the proof shows that the expected total cost of edges whose endpoints are at different vertices of Δ_k in the rounded solution $\bar{x}$ is at most $1.5\,Z^*$. To get an $(1.5 - 1/k)Z^*$ upper bound, one must rename the terminals such that terminal k maximizes a certain quantity given by the simplex relaxation, or alternatively randomly pick a terminal as the last element of the permutation (the order of the first $k-1$ terminals does not matter as long as both the increasing and the decreasing permutations are tried by the rounding procedure). Exhaustive search of the sample space produces one integral solution whose cost does not exceed the average.

Applications

MULTIWAY CUT is used in Computer Vision, but unless one can solve the instance exactly, algorithms for the generalization METRIC LABELING are needed. MULTIWAY CUT has applications in parallel and distributed computing, as well as in chip design.

Open Problems

The improvements of [10,4] are based on better rounding procedures and both compare the integral solution obtained to Z^*. This leads to the natural question: what is the supremum, over multiway cut instances G, of $Z^*(G)/\text{MWC}(G)$. This supremum is called *integrality gap* or *integrality ratio*. For three terminals, [10] and [4] show that the integrality gap is exactly 12/11, while for general k, Freund and Karloff [7] give a lower bound of 8/7. The best-known upper bound is 1.3438, achieved by an approximation algorithm of [10].

Cross References

▶ Multicut
▶ Sparsest Cut

Recommended Reading

1. Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: Network Flows. Prentice Hall, Englewood Cliffs (1993)
2. Arora, S., Karger, D., Karpinski, M.: Polynomial time approximation schemes for dense instances of NP-hard problems. J. Comput. Syst. Sci. **58**(1), 193–210 (1999). Preliminary version in STOC 1995
3. Calinescu, G., Karloff, H.J., Rabani, Y.: An Improved Approximation Algorithm for Multiway Cut. In: ACM Symposium on Theory of Computing 1998, pp. 48–52. Journal version in J. Comp. Syst. Sci. **60**, 564–574 (2000)
4. Cheung, K., Cunningham, W.H., Tang, L.: Optimal 3-Terminal Cuts and Linear Programming. Math. Program. **105**, 389–421 (2006), Preliminary version in IPCO 1999
5. Chopra, S., Rao, M.R.: On the Multiway Cut Polyhedron. Networks **21**, 51–89 (1991)
6. Dahlhaus, E., Johnson, D.S., Papadimitriou, C.H., Seymour, P.D., Yannakakis, M.: The Complexity of Multiterminal Cuts. SIAM J. Comp. **23**, 864–894 (1994). Preliminary version in STOC 1992, An extended abstract was first announced in 1983
7. Freund, A., Karloff, H.: A lower bound of $8/(7+\frac{1}{k-1})$ on the integrality ratio of the Calinescu–Karloff–Rabani relaxation for Multiway Cut. Inf. Process. Lett. **75**, 43–50 (2000)
8. Frieze, A., Kannan, R.: The Regularity Lemma and Approximation Schemes for Dense Problems. In: Proc. 37th IEEE FOCS 1996, pp. 12–20. IEEE Computer Society Press, Los Alamitos
9. Garg, N., Vazirani, V.V., Yannakakis, M.: Multiway cuts in node weighted graphs. J. Algorithms **50**(1), 49–61 (2004). Preliminary version in ICALP 1994
10. Karger, D.R., Klein, P., Stein, C., Thorup, M., Young, N.E.: Rounding algorithms for a geometric embedding of minimum multiway cut. Math. Oper. Res. **29**(3), 436–461 (2004). Preliminary version in STOC 1999
11. Naor, J.S., Zosin, L.: A 2-Approximation Algorithm for the Directed Multiway Cut Problem. SIAM J. Comput. **31**(2), 477–492 (2001). Preliminary version in FOCS 1997
12. Vazirani, V.V.: Approximation Algorithms. Springer, Berlin Heidelberg New York (2001)

N

Nash Equilibria and Dominant Strategies in Routing

2005; Wang, Li, Chu

WEIZHAO WANG[1], XIANG-YANG LI[2], XIAOWEN CHU[3]
[1] Google Inc., Irvine, CA, USA
[2] Department of Computer Science, Illinois Institute of Tech., Chicago, IL, USA
[3] Department of Computer Science, Hong Kong Baptist University, Hong Kong, China

Keywords and Synonyms

Strategyproof; Truthful; Nash; BB

Problem Definition

This problem is concerned with the multicast routing and cost sharing in a selfish network composed of relay terminals and receivers. This problem is motivated by the recent observation that the selfish behavior of the network could largely degraded existing system performance, even dysfunction. The work of Wang, Li and Chu [7] first presented some negative results of the strategyproof mechanism in multicast routing and sharing, and then proposed a new solution based on Nash Equilibrium that could greatly improve the performance.

Wang, Li and Chu modeled a network by a link weighted graph $G = (V, E, \mathbf{c})$, where V is the set of all nodes and $\mathbf{c}$ is the cost vector of the set E of links. For a multicast session, let Q denote the set of all receivers. In game theoretical networking literatures, usually there are two models for the multicast cost/payment sharing.

Axiom Model (AM) All receivers must receive the service, or equivalently, each receiver has an infinity valuation [3]. In this model, a sharing method ξ computes how much each receiver should pay when the receiver set is R and cost vector is $\mathbf{c}$.

Valuation Model (VM) There is a set $Q = \{q_1, q_2, \cdots, q_r\}$ of r possible receivers. Each receiver $q_i \in Q$ has a valuation $\boldsymbol{\eta}_i$ for receiving the service. Let $\boldsymbol{\eta} = (\eta_1, \eta_2, \ldots, \eta_r)$ be the valuation vector and $\boldsymbol{\eta}_R$ be the valuation vector of a set $R \subseteq Q$ of receivers. In this model, they are interested in a sharing mechanism S consisting of a *selection scheme* $\sigma(\boldsymbol{\eta}, \mathbf{c})$ and a *sharing method* $\xi(\boldsymbol{\eta}, \mathbf{c})$. $\sigma_i(\boldsymbol{\eta}, \mathbf{c})$ denotes whether receiver i receives the service or not, and $\xi_i(\boldsymbol{\eta}, \mathbf{c})$ computes how much the receiver q_i should pay for the multicast service. Let $\mathbb{P}(\boldsymbol{\eta}, \mathbf{c})$ be the total payment for providing the service to the receiver set.

In the valuation model, a receiver who is willing to receive the service is not guaranteed to receive the service. For notational simplicity, $\sigma(\boldsymbol{\eta}, \mathbf{c})$ is used to denote the set of actual receivers. Under the Valuation Model, a *fair* sharing according to the following criteria is studied.

- **Budget Balance**: For the receiver set $R = \sigma(\boldsymbol{\eta}, \mathbf{c})$, $\mathbb{P}(\boldsymbol{\eta}, \mathbf{c}) = \sum_{q_i \in Q} \xi_i(\boldsymbol{\eta}, \mathbf{c})$. If $\alpha \cdot \mathbb{P}(\boldsymbol{\eta}, \mathbf{c}) \leq \sum_{i \in R} \xi_i(\boldsymbol{\eta}, \mathbf{c}) \leq \mathbb{P}(\boldsymbol{\eta}, \mathbf{c})$, for some given parameter

```
1: Compute path LCP$(s, q_j, \mathbf{d})$ and set
   $\phi_j = \frac{\omega(B_{mm}(s,q_j,\mathbf{d}),\mathbf{d})}{r}$ for every $q_j \in Q$.
2: Set $\mathcal{O}_i^{DM}(\eta, \mathbf{d}) = 0$ and $\mathcal{P}_i^{DM}(\eta, \mathbf{d}) = 0$ for each link
   $e_i \notin$ LCP$(s, q_j, \mathbf{d})$.
3: for each receiver $q_j$ do
4:   if $\eta_j \geq \phi_j$ then
5:     Receiver $q_j$ is granted the service and charged
       $\xi_j^{DM}(\eta, \mathbf{d})$, set $R = R \cup q_j$.
6:   else
7:     Receiver $q_j$ is not granted the service and is
       charged 0.
8:   end if
9: end for
10: Set $\mathcal{O}_i^{DM}(\eta, \mathbf{d}) = 1$ and $\mathcal{P}_i^{DM}(\eta, \mathbf{d}) = \mathcal{P}_i^{LCPT}(\eta_R^{=\infty}, \mathbf{d})$
    for each link $e_i \in$ LCPT$(R, \mathbf{d})$.
```

Nash Equilibria and Dominant Strategies in Routing, Algorithm 1
The multicast system $\Psi^{DM} = (\mathcal{M}^{DM}, \mathcal{S}^{DM})$ based on multicast tree LCPT

$0 < \alpha \leq 1$, then $S = (\sigma, \xi)$ is called α-budget-balance. If budget balance is not achievable, then a sharing scheme S may need to be α-budget-balance instead of budget balance.

- **No Positive Transfer** (NPT): Any receiver q_i's sharing should not be negative.
- **Free Leaving**: (FR) The potential receivers who do not receive the service should not pay anything.
- **Consumer Sovereignty** (CS): For any receiver q_i, if $\boldsymbol{\eta}_i$ is sufficiently large, then q_i is guaranteed to be an actual receiver.
- **Group-Strategyproof** (GS): Assume that $\boldsymbol{\eta}$ is the valuation vector and $\boldsymbol{\eta}' \neq \boldsymbol{\eta}$. If $\xi_i(\boldsymbol{\eta}', \mathbf{c}) \geq \xi_i(\boldsymbol{\eta}, \mathbf{c})$ for each $q_i \in \boldsymbol{\eta}$, then $\xi_i(\boldsymbol{\eta}', \mathbf{c}) = \xi_i(\boldsymbol{\eta}, \mathbf{c})$.

Notations

The path with the lowest cost between two odes s and t is denoted as $\mathsf{LCP}(s, t, \mathbf{c})$, and its cost is dented as $|\mathsf{LCP}(s, t, \mathbf{c})|$. Given a simple path P in the graph G with cost vector $\mathbf{c}$, the sum of the cost of links on path P is denoted as $|\mathsf{P}(\mathbf{c})|$. For a simple path $P = v_i \rightsquigarrow v_j$, if $\mathsf{LCP}(s, t, \mathbf{c}) \bigcap P = \{v_i, v_j\}$, then P is called a *bridge* over $\mathsf{LCP}(s, t, \mathbf{c})$. This bridge P *covers* link e_k if $e_k \in \mathsf{LCP}(v_i, v_j, \mathbf{c})$. Given a link $e_i \in \mathsf{LCP}(s, t, \mathbf{c})$, the path with the minimum cost that covers e_i is denoted as $\mathrm{B}_{\min}(e_i, \mathbf{c})$. The bridge $\mathrm{B}_{mm}(s, t, \mathbf{c}) = \max_{e_i \in \mathsf{LCP}(s,t,\mathbf{c})} \mathrm{B}_{\min}(e_i, \mathbf{c})$ is the *max-min cover* of the path $\mathsf{LCP}(s, t, \mathbf{c})$.

A bridge set $\mathcal{B}$ is a *bridge cover* for $\mathsf{LCP}(s, t, \mathbf{c})$, if for every link $e_i \in \mathsf{LCP}(s, t, \mathbf{c})$, there exists a bridge $\mathrm{B} \in \mathcal{B}$ such that $e_i \in \mathsf{LCP}(v_{s(\mathrm{B})}, v_{t(\mathrm{B})}, \mathbf{c})$. The *weight* of a bridge cover $\mathcal{B}(s, t, \mathbf{c})$ is defined as $|\mathcal{B}(s, t, \mathbf{c})| = \sum_{B \in \mathcal{B}(s,t,\mathbf{c})} \sum_{e_i \in B} c_i$. A bridge cover is a *least bridge cover* (LB), denoted by $\mathbb{LB}(s, t, \mathbf{c})$, if it has the smallest weight among all bridge covers that cover $\mathsf{LCP}(s, t, \mathbf{c})$.

Key Results

Theorem 1 *If $\Psi = (\mathcal{M}, S)$ is an α-stable multicast system, then $\alpha \leq 1/n$.*

Theorem 2 *Multicast system Ψ^{DM} is $1/(r \cdot n)$-stable, where r is the number of receivers.*

Theorem 1 gives an upper bound for α for any α-stable unicast system Ψ. It is not difficult to observe that even the receivers are cooperative, Theorem 1 still holds. Theorem 2 showed that there exists a multicast system is $1/(r \cdot n)$-stable. When $r = 1$, the problem become traditional unicast system and the bound is tight. When relaxing the dominant strategy to the Nash Equilibria requirement, a First Price Auction (FPA) mechanism is proposed by Wang et al. under the Axiom Model that has many nice properties.

```
1: Each terminal bids a price $b_i$.
2: Every link sends a unit size dummy packet with property $\rho = \tau \cdot (n \cdot b_u - \sum_{e_i \in G} b_i)$ and receives payment
   $f_i(s, q_1, \mathbf{b}) = \tau \cdot \left[ b_u \cdot (n \cdot b_u - \sum_{e_j \in G - e_i} b_j) - \frac{h_i^2}{2} \right]$.
   Here, $b_u$ is the maximum cost any link can declare.
3: Compute the unique path $\mathsf{LCP}(s, q_1, \mathbf{b}')$ by applying certain fixed tie-breaking rule consistently.
4: Each terminal bids again for a price $b_i'$.
5: for each link $e_i$ do
6:     It is select to relay the packet and receives payment $b_i'$ if and only if $e_i$ is on path $\mathsf{LCP}(s, q_1, \mathbf{b}')$.
7: end for
```

Nash Equilibria and Dominant Strategies in Routing, Algorithm 2
FPA Mechanism $\mathcal{M}^{\mathrm{AUC}}$

```
1: Execute Line 1 − 3 in Algorithm 2.
2: Compute $\mathbb{LB}(s, q_1, \mathbf{b})$, and set $\phi = \frac{|\mathbb{LB}(s,q_1,\mathbf{b})|}{2}$.
3: If $\phi \leq \eta_1$ then set $\sigma_1^{\mathrm{AU}}(\eta_1, \widetilde{\mathbf{b}}) = 1$ and $\xi_1^{\mathrm{AU}}(\eta_1, \widetilde{\mathbf{b}}) = \phi$.
   Every relay link on LCP is selected and receives an extra payment $b_i'$.
4: For each link $e_i \notin \mathsf{LCP}(s, q_1, \mathbf{b}')$, it receives a payment $\mathcal{P}_i^{\mathrm{AU}}(\eta_1, \widetilde{\mathbf{b}}) - \gamma \cdot (b_i' - b_i)^2$.
```

Nash Equilibria and Dominant Strategies in Routing, Algorithm 3
FPA based unicast system

Theorem 3 *There exists NE for FPA mechanism $\mathcal{M}^{AUC}$ and for any NE, (a) each link bids his true cost as the first bid b_i, (b) the actual shortest path is always selected, (c) the total cost for different NE differs at most 2 times.*

Based on the FPA Mechanism Ψ^{AUC}, Wang, Li and Chu design a unicast system as follows.

Theorem 4 *The FPA based unicast system not only has Nash Equilibria, but also is $\frac{1}{2}$-NE-stable with ϵ additive, for any given ϵ.*

By treating each receiver as a separate receiver and applying the similar process as in the unicast system, Wang, Li and Chu extended the unicast system to a multicast system.

Theorem 5 *The FPA based multicast system not only has Nash Equilibria, but also is $1/(2 \cdot r)$-NE-stable with ϵ additive, for any given ϵ.*

Applications

More and more research effort has been done to study the non-cooperative games recently. Among these various forms of games, the unicast/multicast routing game [2,5,6] and multicast cost sharing game [1,3,4] have received a considerable amount of attentions over the past few year due to its application in the Internet. However, both unicast/multicast routing game and multicast cost sharing game are one folded: the unicast/multicast routing game does not take the receivers into account while the multicast cost sharing game does not treat the links as non-cooperative. In this paper, they study the scenario, which was called *multicast system*, in which both the links and the receivers could be non-cooperative. Solving this problem paving a way for the real world commercial multicast and unicast application. A few examples are, but not limited to, the multicast of the video content in wireless mesh network and commercial WiFi system; the multicast routing in the core Internet.

Open Problems

A number of problems related to the work of Wang, Li and Chu [7] remain open. The first and foremost, the upper bound and lower bound on α still have a gap of r if the multicast system is α-stable; and a gap of $2r$ if the multicast system is α-Nash stable.

The second, Wang, Li and Chu only showed the existence of the Nash Equilibrium under their systems. They have not characterized the convergence of the Nash Equilibrium and the strategies of the user, which are not only interesting but also important problems.

Cross References

► Non-approximability of Bimatrix Nash Equilibria

Recommended Reading

1. Feigenbaum, J., Papadimitriou, C.H., Shenker, S.: Sharing the cost of multicast transmissions. J. Comput. Syst. Sci. **63**, 21–41 (2001)
2. Kao, M.-Y., Li, X.-Y., Wang, W.: Towards truthful mechanisms for binary demand games: A general framework. In: ACM EC, pp. 213–222, Vancouver, Canada (2005)
3. Herzog, S., Shenker, S., Estrin, D.: Sharing the "cost" of multicast trees: an axiomatic analysis. IEEE/ACM Trans. Netw. **5**, 847–860 (1997)
4. Moulin, H., Shenker, S.: Strategyproof sharing of submodular costs: budget balance versus efficiency. Econ. Theory **18**, 511–533 (2001)
5. Wang, W., Li, X.-Y., Sun, Z., Wang, Y.: Design multicast protocols for non-cooperative networks. In: Proceedings of the 24th IEEE INFOCOM. vol. 3, pp. 1596–1607, Miami, USA (2005)
6. Wang, W., Li, X.-Y., Wang, Y.: Truthful multicast in selfish wireless networks. In: Proceedings of the 10th ACM MOBICOM, pp. 245–259, Philadelphia, USA (2004)
7. Wang, W., Li, X.-Y., Chu, X.: Nash equilibria, dominant strategies in routing. In: Workshop for Internet and Network Economics (WINE). Lecture Notes in Computer Science, vol. 3828, pp 979–988. Springer, Hong Kong, China (2005)

Navigation

► Mobile Agents and Exploration
► Robotics

Nearest Neighbor Interchange and Related Distances

1999; DasGupta, He, Jiang, Li, Tromp, Zhang

BHASKAR DASGUPTA[1], XIN HE[2], TAO JIANG[3], MING LI[4], JOHN TROMP[5], LOUXIN ZHANG[6]
[1] Department of Computer Science, University of Illinois at Chicago, Chicago, IL, USA
[2] Department of Computer Science and Engineering, University at Buffalo The State University of New York, Buffalo, NY, USA
[3] Department of Computer Science and Engineering, University of California at Riverside, Riverside, CA, USA
[4] Department of Computer Science, University of Waterloo, Waterloo, ON, Canada
[5] CWI, Amsterdam, Netherlands
[6] Department of Mathematics, National University of Singapore, Singapore, Singapore

Keywords and Synonyms

Comparison of phylogenies; Network models of evolution

Problem Definition

In this chapter, the authors state results on some transformation based distances for *evolutionary trees*. Several distance models for evolutionary trees have been proposed in the literature. Among them, the best known is perhaps the *nearest neighbor interchange* (nni) distance introduced independently in [10] and [9]. The authors will focus on the nni distance and a closely related distance called the *subtree-transfer* distance originally introduced in [5,6]. Several papers that involved DasGupta, He, Jiang, Li, Tromp and Zhang essentially showed the following results:

- A correspondence between the nni distance and the linear-cost subtree-transfer distance on unweighted trees;
- Computing the nni distance is NP-hard, but admits a fixed-parameter tractability and a logarithmic ratio approximation algorithms;
- A 2-approximation algorithm for the linear-cost subtree-transfer distance on weighted evolutionary trees.

The authors first define the nni and linear-cost subtree-transfer distances for unweighted trees. Then the authors extend the nni and linear-cost subtree-transfer distances to weighted trees. For the purpose of this chapter, an evolutionary tree (also called *phylogeny*) is an *unordered* tree, has uniquely labeled leaves and unlabeled interior nodes, can be *unrooted* or *rooted*, can be *unweighted* or *weighted*, and has all internal nodes of degree 3.

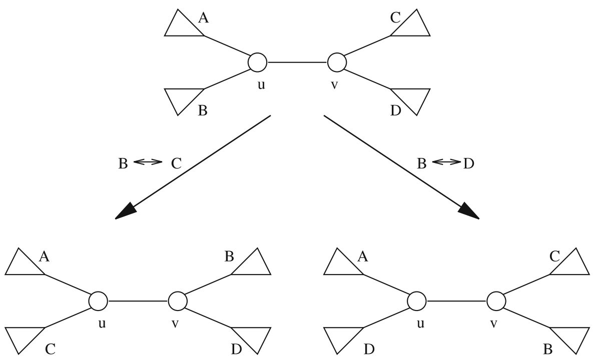
Unweighted Trees

An *nni* operation swaps two subtrees that are separated by an internal edge (u, v), as shown in Fig. 1.

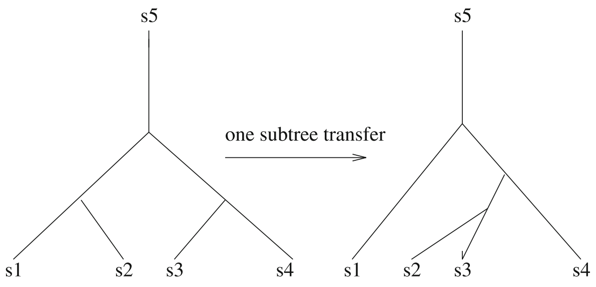
The nni operation is said to *operate* on this internal edge. The nni distance, $D_{\text{nni}}(T_1, T_2)$, between two trees T_1 and T_2 is defined as the *minimum* number of nni operations required to transform one tree into the other.

An nni operation can also be viewed as moving a subtree past a neighboring internal node. A more general operation is to transfer a subtree from one place to another arbitrary place. Figure 2 shows such a *subtree-transfer* operation.

The subtree-transfer distance between two trees T_1 and T_2 is the minimum number of subtrees one need to move to transform T_1 into T_2 [5,6,7]. It is sometimes



Nearest Neighbor Interchange and Related Distances, Figure 1
The two possible nni operations on an internal edge (u, v): exchange $B \leftrightarrow C$ or $B \leftrightarrow D$



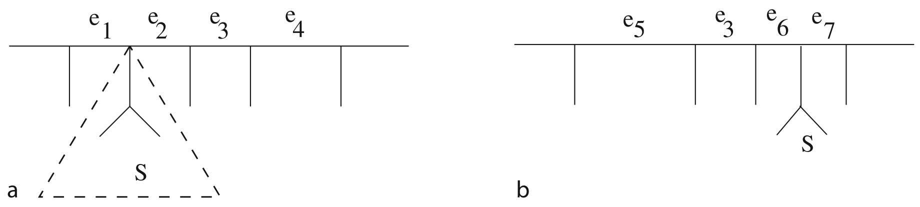
Nearest Neighbor Interchange and Related Distances, Figure 2
An example of subtree-transfer

appropriate in practice to discriminate among subtree-transfer operations as they occur with different frequencies. In this case, one can charge each subtree-transfer operation a cost equal to the distance (the number of nodes passed) that the subtree has moved in the current tree. The *linear-cost* subtree-transfer distance, $D_{\text{lcst}}(T_1, T_2)$, between two trees T_1 and T_2 is then the minimum total cost required to transform T_1 into T_2 by subtree-transfer operations [1,2].

Weighted Trees

Both the linear-cost subtree-transfer and nni models can be naturally extended to weighted trees. The extension for nni is straightforward: an nni operation is simply charged a cost equal to the weight of the edge it operates on. For feasibility of weighted nni transformation between two given weighted trees T_1 and T_2, one also requires that the following conditions are satisfied: (**1**) for each leaf label a, the weight of the edge in T_1 incident on a is the same as the weight of the edge in T_2 incident on a and (**2**) the multisets of weights of internal edges of T_1 and T_2 are the same.

In the case of linear-cost subtree-transfer, although the idea is immediate, i. e., a moving subtree should be charged for the weighted distance it travels, the formal definition needs some care and is given below. Consider (unrooted) trees in which each edge e has a weight $w(e) \geq 0$. To ensure feasibility of transforming a tree into another, One requires the total weight of all edges to equal one. A subtree-transfer is now defined as follows. Select a subtree S of T at a given node u and select an edge $e \notin S$. Split the edge e into two edges e_1 and e_2 with weights $w(e_1)$ and $w(e_2)$ ($w(e_1), w(e_2) \geq 0, w(e_1) + w(e_2) = w(e)$), and move S to the common end-point of e_1 and e_2. Finally, merge the two remaining edges e' and e'' adjacent to u into one edge with weight $w(e') + w(e'')$. The cost of this subtree-transfer is the total weight of all the edges over which S



Nearest Neighbor Interchange and Related Distances, Figure 3
Subtree-transfer on weighted phylogenies. Tree b is obtained from tree a with one subtree-transfer

is moved. Figure 1 gives an example. The edge-weights of the given tree are normalized so that their total sum is 1. The subtree S is transferred to split the edge e_4 to e_6 and e_7 such that $w(e_6), w(e_7) \geq 0$ and $w(e_6) + w(e_7) = w(e_4)$; finally, the two edges e_1 and e_2 are merged to e_5 such that $w(e_5) = w(e_1) + w(e_2)$. The cost of transferring S is $w(e_2) + w(e_3) + w(e_6)$.

Note that for weighted trees, the linear-cost subtree-transfer model is more general than the nni model in the sense that one can slide a subtree along an edge with subtree-transfers. Such an operation is not realizable with nni moves.

Key Results

Let T_1 and T_2 be the two trees, each with n nodes, that are being used in the distance computation.

Theorem 1 ([2,3,4]) *Assume that T_1 and T_2 are unweighted. Then, the following results hold:*

- $D_{nni}(T_1, T_2) = D_{lcst}(T_1, T_2)$.
- *Computing $D_{nni}(T_1, T_2)$ is NP-complete.*
- *Suppose that $D_{nni}(T_1, T_2) \leq d$. Then, an optimal sequence of nni operations transforming T_1 into T_2 can be computed in $O(n^2 \log n + n \cdot 2^{23d/2})$ time.*
- *$D_{nni}(T_1, T_2)$ can be approximated to within a factor of $\log n + O(1)$ in polynomial time.*

Theorem 2 ([1,2,3,4]) *Assume that T_1 and T_2 are weighted. Then, the following results hold:*

- *$D_{nni}(T_1, T_2)$ can be approximated to within a factor of $6 + 6 \log n$ in $O(n^2 \log n)$ time.*
- *Assume that T_1 and T_2 are allowed to have leaves that are not necessarily uniquely labeled. Then, computing $D_{lcst}(T_1, T_2)$ is NP-hard.*
- *$D_{lcst}(T_1, T_2)$ can be approximated to within a factor of 2 in $O(n^2 \log n)$ time.*

Applications

The results reported here are on transformation based distances for evolutionary trees. Such a tree is can be *rooted* if the evolutionary origin is known and can be *weighted* if the evolutionary length on each edge is known. Reconstructing the correct evolutionary tree for a set of species is one of the fundamental yet difficult problems in evolutionary genetics. Over the past few decades, many approaches for reconstructing evolutionary trees have been developed, including (not exhaustively) parsimony, compatibility, distance and maximum likelihood approaches. The outcomes of these methods usually depend on the data and the amount of computational resources applied. As a result, in practice they often lead to different trees on the same set of species [8]. It is thus of interest to compare evolutionary trees produced by different methods, or by the same method on different data.

Another motivation for investigating the linear-cost subtree transfer distance comes from the following motivation. When *recombination* of DNA sequences occurs in an evolution, two sequences meet and generate a new sequence, consisting of genetic material taken left of the recombination point from the first sequence and right of the point from the second sequence [5,6]. From a phylogenetic viewpoint, before the recombination, the ancestral material on the present sequence was located on two sequences, one having all the material to the left of the recombination point and another having all the material to the right of the breaking point. As a result, the evolutionary history can no longer be described by a single tree. The recombination event partitions the sequences into two neighboring regions. The history for the left and the right regions could be described by separate evolutionary trees. The recombination makes the two evolutionary trees describing neighboring regions differ. However, two neighbor trees cannot be arbitrarily different, one must be obtainable from the other by a *subtree-transfer operation*. When more than one recombination occurs, one can describe an evolutionary history using a list of evolutionary trees, each corresponds to some region of the sequences and each can be obtained by several subtree-transfer operations from its predecessor [6]. The computation of a linear-cost subtree-transfer distance is useful in reconstructing such a list of trees based on parsimony [5,6].

Open Problems

1. Is there a constant ratio approximation algorithm for the nni distance on unweighted evolutionary trees or is the $O(\log n)$-approximation the best possible?
2. Is the linear-cost subtree-transfer distance NP-hard to compute on weighted evolutionary trees if leaf labels are not allowed to be non-unique?
3. Can one improve the approximation ratio for linear-cost subtree-transfer distance on weighted evolutionary trees?

Cross References

▶ Constructing a Galled Phylogenetic Network
▶ Maximum Agreement Subtree (of 2 Binary Trees)
▶ Maximum Agreement Subtree (of 3 or More Trees)
▶ Phylogenetic Tree Construction from a Distance Matrix

Recommended Reading

1. DasGupta, B., He, X., Jiang, T., Li, M., Tromp, J.: On the linear-cost subtree-transfer distance. Algorithmica **25**(2), 176–195 (1999)
2. DasGupta, B., He, X., Jiang, T., Li, M., Tromp, J., Zhang, L.: On distances between phylogenetic trees, 8th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 427–436 (1997)
3. DasGupta, B., He, X., Jiang, T., Li, M., Tromp, J., Wang, L., Zhang, L.: Computing Distances between Evolutionary Trees. In: Du, D.Z., Pardalos, P.M. (eds.) Handbook of Combinatorial Optimization. Kluwer Academic Publishers, Norwell, **2**, 35–76 (1998)
4. DasGupta, B., He, X., Jiang, T., Li, M., Tromp, J., Zhang, L.: On Computing the Nearest Neighbor Interchange Distance. In: Du, D.Z., Pardalos, P.M., Wang, J. (eds.) Proceedings of the DIMACS Workshop on Discrete Problems with Medical Applications, DIMACS Series in Discrete Mathematics and Theoretical Computer Science. Am. Math. Soc. **55**, 125–143 (2000)
5. Hein, J.: Reconstructing evolution of sequences subject to recombination using parsimony. Math. Biosci. **98**, 185–200 (1990)
6. Hein, J.: A heuristic method to reconstruct the history of sequences subject to recombination. J. Mol. Evol. **36**, 396–405 (1993)
7. Hein, J., Jiang, T., Wang, L., Zhang, K.: On the complexity of comparing evolutionary trees. Discret. Appl. Math. **71**, 153–169 (1996)
8. Kuhner, M., Felsenstein, J.: A simulation comparison of phylogeny algorithms under equal and unequal evolutionary rates. Mol. Biol. Evol. **11**(3), 459–468 (1994)
9. Moore, G.W., Goodman, M., Barnabas, J.: An iterative approach from the standpoint of the additive hypothesis to the dendrogram problem posed by molecular data sets. J. Theor. Biol. **38**, 423–457 (1973)
10. Robinson, D.F.: Comparison of labeled trees with valency three. J Combinator. Theory Series B **11**, 105–119 (1971)

Negative Cycles in Weighted Digraphs

1994; Kavvadias, Pantziou, Spirakis, Zaroliagis

CHRISTOS ZAROLIAGIS
Computer Engineering & Informatics,
University of Patras, Patras, Greece

Problem Definition

Let $G = (V, E)$ be an n-vertex, m-edge directed graph (digraph), whose edges are associated with a real-valued cost function $wt : E \to \mathbb{R}$. The cost, $wt(P)$, of a path P in G is the sum of the costs of the edges of P. A simple path C whose starting and ending vertices coincide is called a cycle. If $wt(C) < 0$, then C is called a *negative cycle*. The goal of the negative cycle problem is to detect whether there is such a cycle in a given digraph G with real-valued edge costs, and if indeed exists to output the cycle.

The negative cycle problem is closely related to the shortest path problem. In the latter, a minimum cost path between two vertices s and t is sought. It is easy to see that an s-t shortest path exists if and only if no s-t path in G contains a negative cycle [1,13]. It is also well-known that shortest paths from a given vertex s to all other vertices form a tree called *shortest path tree* [1,13].

Key Results

For the case of general digraphs, the best algorithm to solve the negative cycle problem (or to compute the shortest path tree, if such a cycle does not exist) is the classical Bellman–Ford algorithm that takes $O(nm)$ time (see e. g., [1]). Alternative methods with the same time complexity are given in [4,7,12,13]. Moreover, in [11, Chap. 7] an extension of the Bellman–Ford algorithm is described which, in addition to detecting and reporting the existing negative cycles (if any), builds a shortest path tree rooted a some vertex s reaching those vertices u whose shortest s-u path does not contain a negative cycle. If edge costs are integers larger than $-L$ ($L \geq 2$), then a better algorithm was given in [6] that runs in $O(m\sqrt{n}\log L)$ time, and it is based on bit scaling.

A simple deterministic algorithm that runs in $O(n^2 \log n)$ expected time with high probability is given in [10] for a large class of input distributions, where the edge costs are chosen randomly according to the endpoint-independent model (this model includes the common case where all edge costs are chosen independently from the same distribution).

Better results are known for several important classes of sparse digraphs (i. e., digraphs with $m = O(n)$ edges) such as planar digraphs, outerplanar digraphs, digraphs of small genus, and digraphs of small treewidth.

For general sparse digraphs, an algorithm is given in [8] that solves the negative cycle problem in $O(n + \tilde{\gamma}^{1.5} \log \tilde{\gamma})$ time, where $\tilde{\gamma}$ is a topological measure of the input sparse digraph G, and whose value varies from 1 up to $\Theta(n)$. Informally, $\tilde{\gamma}$ represents the minimum number of outerplanar subgraphs, satisfying certain separation properties, into which G can be decomposed. In particular, $\tilde{\gamma}$ is proportional to $\gamma(G) + q$, where G is supposed to be embedded into an orientable surface of genus $\gamma(G)$ so as to minimize the number q of faces that collectively cover all vertices. For instance, if G is outerplanar, then $\tilde{\gamma} = 1$, which implies an optimal $O(n)$ time algorithm for this case. The algorithm in [8] does not require such an embedding to be provided by the input. In the same paper, it is shown that random $G_{n,p}$ graphs with threshold function $1/n$ are planar with probability one and have an expected value for $\tilde{\gamma}$ equal to $O(1)$. Furthermore, an efficient parallelization of the algorithm on the CREW PRAM model of computation is provided in [8].

Better bounds for planar digraphs are as follows. If edge costs are integers, then an algorithm running in $O(n^{4/3} \log(nL))$ time is given in [9]. For real edge costs, an $O(n \log^3 n)$-time algorithm was given in [5].

An optimal $O(n)$-time algorithm is given in [3] for the case of digraphs with small treewidth (and real edge costs). Informally, the treewidth t of a graph G is a parameter which measures how close is the structure of G to a tree. For instance, the class of graphs of small treewidth includes series-parallel graphs ($t = 2$) and outerplanar graphs ($t = 2$). An optimal parallel algorithm for the same problem, on the EREW PRAM model of computation, is provided in [2].

Applications

Finding negative cycles in a digraph is a fundamental combinatorial and network optimization problem that spans a wide range of applications including: shortest path computation, two dimensional package element, minimum cost flows, minimal cost-to-time ratio, model verification, compiler construction, software engineering, VLSI design, scheduling, circuit production, constraint programming and image processing. For instance, the isolation of negative feedback loops is imperative in the design of VLSI circuits. It turns out that such loops correspond to negative cost cycles in the so-called amplifier-gain graph of the circuit. In constraint programming, it is required to check the feasibility of sets of constraints. Systems of difference constraints can be represented by constraint graphs, and one can show that such a system is feasible if and only if there are no negative cost cycles in its corresponding constraint graph. In zero-clairvoyant scheduling, the problem of checking whether there is a valid schedule in such a scheduling system can be reduced to detecting negative cycles in an appropriately defined graph. For further discussion on these and other applications see [1,12,14].

Open Problems

The negative cycle problem is closely related to the shortest path problem. The existence of negative edge costs makes the solution of the negative cycle problem or the computation of a shortest path tree more difficult and thus more time consuming compared to the time required to solve the shortest path tree problem in digraphs with non-negative edge costs. For instance, for digraphs with real edge costs, compare the $O(nm)$-time algorithm in the former case with the $O(m + n \log n)$-time algorithm for the latter case (Dijkstra's algorithm implemented with an efficient priority queue; see e. g., [1]).

It would therefore be interesting to try to reduce the gap between the above two time complexities, even for special classes of graphs or the case of integer costs.

The only case where these two complexities coincide concerns the digraphs of small treewidth [3], making it the currently most general such class of graphs. For planar digraphs, the result in [5] is only a polylogarithmic factor away from the $O(n)$-time algorithm in [9] that computes a shortest path tree when the edge costs are non-negative.

Experimental Results

An experimental study for the negative cycle problem is conducted in [4]. In that paper, several methods that combine a shortest path algorithm (based on the Bellman–Ford approach) with a cycle detection strategy are investigated, along with some new variations of them. It turned out that the performance of algorithms for the negative cycle problem depends on the number and the size of the negative cycles. This gives rise to a collection of problem families for testing negative cycle algorithms.

A follow-up of the above study is presented in [14], where two new heuristics are introduced and are incorporated on three of the algorithms considered in [4] (the original Bellman–Ford and the variations in [13] and [7]), achieving dramatic improvements. The data sets considered in [14] are those in [4].

Data Sets

Data set generators and problem families are described in [4], and are available from http://www.avglab.com/andrew/soft.html.

URL to Code

The code used in [4] is available from http://www.avglab.com/andrew/soft.html.

Cross References

► All Pairs Shortest Paths in Sparse Graphs
► All Pairs Shortest Paths via Matrix Multiplication
► Single-Source Shortest Paths

Recommended Reading

1. Ahuja, R., Magnanti, T., Orlin, J.: Network Flows. Prentice-Hall, Englewood Cliffs (1993)
2. Chaudhuri, S., Zaroliagis, C.: Shortest Paths in Digraphs of Small Treewidth. Part II: Optimal Parallel Algorithms. Theor. Comput. Sci. **203**(2), pp. 205–223 (1998)
3. Chaudhuri, S., Zaroliagis, C.: Shortest Paths in Digraphs of Small Treewidth. Part I: Sequential Algorithms. Algorithmca **27**(3), pp. 212–226 (2000)
4. Cherkassky, B.V., Goldberg, A.V.: Negative-Cycle Detection Algorithms. Math. Program. **85**, pp. 277–311 (1999)
5. Fakcharoenphol, J., Rao, S.: Planar Graphs, Negative Weight Edges, Shortest Paths, and near Linear Time. In: Proc. 42nd IEEE Symp. on Foundations of Computer Science – FOCS (2001), pp. 232–241. IEEE Computer Society Press, Los Alamitos (2001)
6. Goldberg, A.V.: Scaling Algorithms for the Shortest Paths Problem. SIAM J. Comput. **24**, pp. 494–504 (1995)
7. Goldberg, A.V., Radzik, T.: A Heuristic Improvement of the Bellman—Ford Algorithm. Appl. Math. Lett. **6**(3), pp. 3–6 (1993)
8. Kavvadias, D., Pantziou, G., Spirakis, P., Zaroliagis, C.: Efficient Sequential and Parallel Algorithms for the Negative Cycle Problem. In: Algorithms and Computation – ISAAC'94. Lect. Notes Comput. Sci., vol. 834, pp.270–278. Springer, Heidelberg (1994)
9. Klein, P., Rao, S., Rauch, M., Subramanian, S.: Faster shortest-path algorithms for planar graphs. J. Comput. Syst. Sci. **5**(1), pp. 3–23 (1997)
10. Kolliopoulos, S.G., Stein, C.: Finding Real-Valued Single-Source Shortest Paths in $o(n^3)$ Expected Time. J. Algorithms **28**, pp. 125–141 (1998)
11. Mehlhorn, K., Näher, S.: LEDA: A Platform for Combinatorial and Geometric Computing. Cambridge University Press, Cambridge (1999)
12. Spirakis, P., Tsakalidis, A.: A Very Fast, Practical Algorithm for Finding a Negative Cycle in a Digraph. In Proc. of 13th ICALP, pp. 397–406 (1986)
13. Tarjan, R.E.: Data Structures and Network Algorithms. SIAM, Philadelphia (1983)
14. Wong, C.H., Tam, Y.C.: Negative Cycle Detection Problem. In: Algorithms – ESA 2005. Lecture Notes in Computer Science, vol. 3669, pp. 652–663. Springer, Heidelberg (2005)

Non-approximability of Bimatrix Nash Equilibria

2006; Chen, Deng, Teng

XI CHEN[1], XIAOTIE DENG[2]
[1] Computer Science and Technology, Tsinghua University, Beijing, Beijing, China
[2] Department of Computer Science, City University of Hong Kong, Hong Kong, China

Keywords and Synonyms

Approximate Nash equilibrium

Problem Definition

In this entry, the following two problems are considered: 1) the problem of finding an approximate Nash equilibrium in a positively normalized bimatrix (or two-player) game; and 2) the smoothed complexity of finding an exact Nash equilibrium in a bimatrix game. It turns out that these two problems are strongly correlated [3].

Let $\mathcal{G} = (\mathbf{A}, \mathbf{B})$ be a bimatrix game, where $\mathbf{A} = (a_{i,j})$ and $\mathbf{B} = (b_{i,j})$ are both $n \times n$ matrices. Game $\mathcal{G}$ is said to be positively normalized, if $0 \le a_{i,j}, b_{i,j} \le 1$ for all $1 \le i, j \le n$.

Let $\mathbb{P}^n$ denote the set of all probability vectors in $\mathbb{R}^n$, i. e., non-negative vectors whose entries sum to 1. A Nash equilibrium [8] of $\mathcal{G} = (\mathbf{A}, \mathbf{B})$ is a pair of mixed strategies $(\mathbf{x}^* \in \mathbb{P}^n, \mathbf{y}^* \in \mathbb{P}^n)$ such that for all $\mathbf{x}, \mathbf{y} \in \mathbb{P}^n$,

$$(\mathbf{x}^*)^{\mathrm{T}}\mathbf{A}\mathbf{y}^* \ge \mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{y}^* \quad \text{and} \quad (\mathbf{x}^*)^{\mathrm{T}}\mathbf{B}\mathbf{y}^* \ge (\mathbf{x}^*)^{\mathrm{T}}\mathbf{B}\mathbf{y}\,,$$

while an ϵ-approximate Nash equilibrium is a pair $(\mathbf{x}^* \in \mathbb{P}^n, \mathbf{y}^* \in \mathbb{P}^n)$ that satisfies

$$\begin{aligned}&(\mathbf{x}^*)^{\mathrm{T}}\mathbf{A}\mathbf{y}^* \ge \mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{y}^* - \epsilon \quad \text{and}\\ &(\mathbf{x}^*)^{\mathrm{T}}\mathbf{B}\mathbf{y}^* \ge (\mathbf{x}^*)^{\mathrm{T}}\mathbf{B}\mathbf{y} - \epsilon\,, \quad \text{for all} \quad \mathbf{x}, \mathbf{y} \in \mathbb{P}^n\,.\end{aligned}$$

In the smoothed analysis [11] of bimatrix games, a perturbation of magnitude $\sigma > 0$ is first applied to the input game: For a positively normalized $n \times n$ game $\mathcal{G} = (\overline{\mathbf{A}}, \overline{\mathbf{B}})$, let $\mathbf{A}$ and $\mathbf{B}$ be two matrices with

$$a_{i,j} = \overline{a}_{i,j} + r^A_{i,j} \quad \text{and} \quad b_{i,j} = \overline{b}_{i,j} + r^B_{i,j}, \quad \forall 1 \le i, j \le n,$$

while $r^A_{i,j}$ and $r^B_{i,j}$ are chosen independently and uniformly from interval $[-\sigma, \sigma]$ or from Gaussian distribution with variance σ^2. These two kinds of perturbations are referred to as *σ-uniform* and *σ-Gaussian perturbations*, respectively. An algorithm for bimatrix games has *polynomial smoothed complexity* (under σ-uniform or σ-Gaussian perturbations) [11], if it finds a Nash equilibrium of game $(\mathbf{A}, \mathbf{B})$ in expected time poly$(n, 1/\sigma)$, for all $(\overline{\mathbf{A}}, \overline{\mathbf{B}})$.

Key Results

The complexity class **PPAD** [9] is defined in entry ▶ Bimatrix Nash. The following theorems are proved in [3].

Theorem 1 *For any constant $c > 0$, the problem of computing a $1/n^c$-approximate Nash equilibrium of a positively normalized $n \times n$ bimatrix game is* **PPAD***-complete.*

Theorem 2 *The problem of computing a Nash equilibrium in a bimatrix game is not in smoothed polynomial time, under uniform or Gaussian perturbations, unless* **PPAD** $\subseteq$ **RP**.

Corollary 1 *The smoothed complexity of the Lemke-Howson algorithm is not polynomial, under uniform or Gaussian perturbations, unless* **PPAD** $\subseteq$ **RP**.

Applications

See entry ▶ Bimatrix Nash.

Open Problems

There remains a complexity gap on the approximation of Nash equilibria in bimatrix games: The result of [7] shows that, an ϵ-approximate Nash equilibrium can be computed in $n^{O(\log n/\epsilon^2)}$-time, while [3] show that no algorithm can find an ϵ-approximate Nash equilibrium in $\text{poly}(n, 1/\epsilon)$-time for ϵ of order $1/\text{poly}(n)$, unless **PPAD** is in **P**. However, the hardness result of [3] does not cover the case when ϵ is a constant between 0 and 1. Naturally, it is unlikely that the problem of finding an ϵ-approximate Nash equilibrium is **PPAD**-complete when ϵ is an absolute constant, for otherwise, all the search problems in **PPAD** would be solvable in $n^{O(\log n)}$-time, due to the result of [7]. An interesting open problem is that, for every constant $\epsilon > 0$, is there a polynomial-time algorithm for finding an ϵ-approximate Nash equilibrium? The following conjectures are proposed in [3]:

Conjecture 1 *There is an $O(n^{k+\epsilon^{-c}})$-time algorithm for finding an ϵ-approximate Nash equilibrium in a bimatrix game, for some constants c and k.*

Conjecture 2 *There is an algorithm to find a Nash equilibrium in a bimatrix game with smoothed complexity $O(n^{k+\sigma^{-c}})$ under perturbations with magnitude σ, for some constants c and k.*

It is also conjectured in [3] that Corollary 1 remains true without any complexity assumption on class **PPAD**. A positive answer would extend the result of [10] to the smoothed analysis framework.

Cross References

▶ Complexity of Bimatrix Nash Equilibria
▶ General Equilibrium
▶ Leontief Economy Equilibrium

Recommended Reading

1. Chen, X., Deng, X.: 3-Nash is PPAD-complete. ECCC, TR05-134 (2005)
2. Chen, X., Deng, X.: Settling the complexity of two-player Nash equilibrium. In: FOCS'06: Proc. of the 47th Annual IEEE Symposium on Foundations of Computer Science, 2006, pp. 261–272
3. Chen, X., Deng, X., Teng, S.H.: Computing Nash equilibria: approximation and smoothed complexity. In: FOCS'06: Proc. of the 47th Annual IEEE Symposium on Foundations of Computer Science, 2006, pp. 603–612
4. Daskalakis, C., Goldberg, P.W., Papadimitriou, C.H.: The complexity of computing a Nash equilibrium. In: STOC'06: Proceedings of the 38th ACM Symposium on Theory of Computing, 2006, pp. 71–78
5. Daskalakis, C., Papadimitriou, C.H.: Three-player games are hard. ECCC, TR05-139 (2005)
6. Goldberg, P.W., Papadimitriou, C.H.: Reducibility among equilibrium problems. In: STOC'06: Proc. of the 38th ACM Symposium on Theory of Computing, 2006, pp. 61–70
7. Lipton, R., Markakis, E., Mehta, A.: Playing large games using simple strategies. In: Proc. of the 4th ACM conference on Electronic commerce, 2003, pp. 36–41
8. Nash, J.F.: Equilibrium point in n-person games. Proc. Natl. Acad. Sci. USA **36**(1), 48–49 (1950)
9. Papadimitriou, C.H.: On the complexity of the parity argument and other inefficient proofs of existence. J. Comput. Syst. Sci. **48**, 498–532 (1994)
10. Savani, R., von Stengel, B.: Exponentially many steps for finding a Nash equilibrium in a bimatrix game. In: FOCS'04: Proc. of the 45th Annual IEEE Symposium on Foundations of Computer Science, 2004, pp. 258–267
11. Spielman, D.A., Teng, S.H.: Smoothed analysis of algorithms and heuristics: progress and open questions. In: Pardo, L.M., Pinkus, A., Süli, E., Todd, M.J. (eds.) Foundations of Computational Mathematics, pp. 274–342. Cambridge University Press, Cambridge, UK (2006)

Non-shared Edges

1985; Day

WING-KAI HON
Department of Computer Science, National Tsing Hua University, Hsin Chu, Taiwan

Keywords and Synonyms

Robinson–Foulds distance; Robinson–Foulds metric

Problem Definition

Phylogenies are binary trees whose leaves are labeled with distinct leaf labels. This problem in this article is concerned with a well-known measurement, called *non-shared edge distance*, for comparing the dissimilarity between two phylogenies. Roughly speaking, the non-shared edge distance counts the number of edges that differentiate one phylogeny from the other.

Let e be an edge in a phylogeny T. Removing e from T splits T into two subtrees. The leaf labels are partitioned into two subsets according to the subtrees. The edge e is said to *induce* a partition of the set of leaf labels. Given two phylogenies T and T' having the same number of leaves with the same set of leaf labels, an edge e in T is *shared* if there exists some edge e' in T' such that the edges e and e' induce the same partition of the set of leaf labels in their corresponding tree. Otherwise, e is *non-shared*. Notice that T and T' has the same number of edges, so that the number of non-shared edges in T (with respect to T') is the same as the number of non-shared edges in T' (with respect to T). Such a number is called the *non-shared edge distance* between T and T'. Two problems are defined as follows:

Non-shared Edge Distance Problem
INPUT: Two phylogenies on the same set of leaf labels
OUTPUT: The non-shared edge distance between the two input phylogenies

All-Pairs Non-shared Edge Distance Problem
INPUT: A collection of phylogenies on the same set of leaf labels
OUTPUT: The non-shared edge distance between each pair of the input phylogenies

Extension

Phylogenies that are commonly used in practice have weights associated to the edges. The notion of non-shared edge can be easily extended for edge-weighted phylogenies. In this case, an edge e will induce a partition of the set of leaf labels as well as the multi-set of edge weights (here, edge weights are allowed to be non-distinct). Given two edge-weighted phylogenies R and R' having the same set of leaf labels and the same multi-set of edge weights, an edge e in R is *shared* if there exists some edge e' in R' such that the edges e and e' induce the same partition of the set of leaf labels and the multi-set of edge weights. Otherwise, e is *non-shared*. The *non-shared edge distance* between R and R' are similarly defined, giving the following problem:

General Non-shared Edge Distance Problem
INPUT: Two edge-weighted phylogenies on the same set of leaf labels and same multi-set of edge weights
OUTPUT: The non-shared edge distance between the two input phylogenies

Key Results

Day [3] proposed the first linear-time algorithm for the Non-shared Edge Distance Problem.

Theorem 1 *Let T and T' be two input phylogenies with the same set of leaf labels, and n be the number of leaves in each phylogeny. The non-shared edge distance between T and T' can be computed in $O(n)$ time.*

Let Δ be a collection of k phylogenies on the same set of leaf labels, and n be the number of leaves in each phylogeny. The All-Pairs Non-shared Edge Distance Problem can be solved by applying Theorem 1 on each pair of phylogenies, thus solving the problem in a total of $O(k^2 n)$ time. Pattengale and Moret [9] proposed a randomized result based on [7] to solve the problem approximately, whose running time is faster when $n \le k \le 2^n$.

Theorem 2 *Let ε be a parameter with $\varepsilon > 0$. Then, there exists a randomized algorithm such that with probability at least $1 - k^{-2}$, the non-shared edge distance between each pair of phylogenies in Δ can be approximated within a factor of $(1+\varepsilon)$ from the actual distance; the running time of the algorithm is $O(k(n^2 + k \log k)/\varepsilon^2)$.*

For general phylogenies, let R and R' be two input phylogenies with the same set of leaf labels and the same multi-set of edge weights, and n be the number of leaves in each phylogeny. The General Non-shared Distance Problem can be solved easily in $O(n^2)$ time by applying Theorem 1 in a straightforward manner. The running time is improved by Hon et al. in [5].

Theorem 3 *The non-shared edge distance between R and R' can be computed in $O(n \log n)$ time.*

Applications

Phylogenies are commonly used by biologists to model the evolutionary relationship among species. Many reconstruction methods (such as maximum parsimony, maximum likelihood, compatibility, distance matrix) produce different phylogenies based on the same set of species, and it is interesting to compute the dissimilarities between them. Also, through the comparison, information about rare genetic events such as recombinations or gene conversions may be uncovered. The most common dissimilarity measure is the Robinson–Foulds metric [11], which is exactly the same as the non-shared edge distance.

Other dissimilarity measures, such as the nearest-neighbor interchange (NNI) distance and the subtree-transfer (STT) distance (see [2] for details), are also proposed in the literature. These measures are sometimes preferred by the biologists since they can be used to deduce the biological events that create the dissimilarity. Nevertheless, these measures are usually difficult to compute. In particular, computing the NNI distance and the STT distance are shown to be NP-hard by DasGupta et al. [1,2]. Approximation algorithms are devised for these problems (NNI: [4,8], STT: [1,6]). Interestingly, all these algorithms make use of the non-shared edge distance to bound their approximation ratios.

Cross References

A related problem is to measure the *similarity* between two input phylogenies. ▶ Maximum Agreement Subtree for references.

Recommended Reading

1. DasGupta, B., He, X., Jiang, T., Li, M., Tromp, J.: On the Linear-Cost Subtree-Transfer Distance between Phylogenetic Trees. Algorithmica **25**(2–3), 176–195 (1999)
2. DasGupta, B., He, X., Jiang, T., Li, M., Tromp, J., Zhang, L.: On Distances between Phylogenetic Trees. In: Proceedings of the Eighth ACM-SIAM Annual Symposium on Discrete Algorithms (SODA), New Orleans, pp. 427–436. SIAM, Philadelphia (1997)
3. Day, W.H.E.: Optimal Algorithms for Comparing Trees with Labeled Leaves. J. Classif. **2**, 7–28 (1985)
4. Hon, W.K., Lam, T.W.: Approximating the Nearest Neighbor Intercharge Distance for Non-Uniform-Degree Evolutionary Trees. Int. J. Found. Comp. Sci. **12**(4), 533–550 (2001)
5. Hon, W.K., Kao, M.Y., Lam, T.W., Sung, W.K., Yiu, S.M.: Non-shared Edges and Nearest Neighbor Interchanges revisited. Inf. Process. Lett. **91**(3), 129–134 (2004)
6. Hon, W.K., Lam, T.W., Yiu, S.M., Kao, M.Y., Sung, W.K.: Subtree Transfer Distance For Degree-D Phylogenies. Int. J. Found. Comp. Sci. **15**(6), 893–909 (2004)
7. Johnson, W., Lindenstrauss, J.: Extensions of Lipschitz Mappings into a Hilbert Space. Contemp. Math. **26**, 189–206 (1984)
8. Li, M., Tromp, J., Zhang, L.: Some Notes on the Nearest Neighbour Interchange Distance. J. Theor. Biol. **26**(182), 463–467 (1996)
9. Pattengale, N.D., Moret, B.M.E.: A Sublinear-Time Randomized Approximation Scheme for the Robinson–Foulds Metric. In: Proceedings of the Tenth ACM Annual International Conference on Research in Computational Molecular Biology (RECOMB), pp. 221–230. Venice, Italy, April 2–5 2006
10. Robinson, D.F.: Comparison of Labeled Trees with Valency Three. J. Comb. Theor. **11**, 105–119 (1971)
11. Robinson, D.F., Foulds, L.R.: Comparison of Phylogenetic Trees. Math. Biosci. **53**, 131–147 (1981)

Nucleolus

2006; Deng, Fang, Sun

QIZHI FANG
Department of Mathematics, Ocean University of China, Qingdao, China

Keywords and Synonyms

Nucleon; Kernel

Problem Definition

Cooperative game theory considers how to distribute the total income generated by a set of participants in a joint project to individuals. The Nucleolus, trying to capture the intuition of minimizing dissatisfaction of players, is one of the most well-known solution concepts among various attempts to obtain a unique solution. In Deng, Fang and Sun's work [2], they study the Nucleolus of flow games from the algorithmic point of view. It is shown that, for a flow game defined on a simple network (arc capacity being all equal), computing the Nucleolus can be done in polynomial time, and for flow games in general cases, both the computation and the recognition of the Nucleolus are $\mathcal{NP}$-hard.

A cooperative (profit) game (N, v) consists of a player set $N = \{1, 2, \cdots, n\}$ and a characteristic function $v : 2^N \to R$ with $v(\emptyset) = 0$, where the value $v(S)$ $(S \subseteq N)$ is interpreted as the profit achieved by the collective action of players in S. Any vector $x \in R^n$ with $\sum_{i \in N} x_i = v(N)$ is an allocation. An allocation x is called an *imputation* if $x_i \geq v(\{i\})$ for all $i \in N$. Denote by $\mathcal{I}(v)$ the set of imputations of the game.

Given an allocation x, the *excess* of a coalition $S (S \subseteq N)$ at x is defined as

$$e(S, x) = x(S) - v(S) ,$$

where $x(S) = \sum_{i \in S} x_i$ for $S \subseteq N$. The value $e(S, x)$ can be interpreted as a measure of satisfaction of coalition S with the allocation x. The core of the game (N, v), denoted by $C(v)$, is the set of allocations whose excesses are all non-negative. For an allocation x of the game (N, v), let $\theta(x)$ denote the $(2^n - 2)$-dimensional vector whose components are the non-trivial excesses $e(S, x)$, $\emptyset \neq S \neq N$, arranged in a non-decreasing order. That is, $\theta_i(x) \leq \theta_j(x)$, for $1 \leq i < j \leq 2^n - 2$. Denote by $\succeq_l$ the "lexicographically greater than" relationship between vectors of the same dimension.

Definition 1 The Nucleolus $\eta(v)$ of game (N, v) is the set of imputations that lexicographically maximize $\theta(x)$ over the set of all imputations $x \in \mathcal{I}(v)$. That is,

$$\eta(v) = \{x \in \mathcal{I}(v) : \theta(x) \succeq_l \theta(y) \text{ for all } y \in \mathcal{I}(v)\}\ .$$

Even though, the Nucleolus may contain multiple points by the definition, it was proved by Schmeidler [12] that the Nucleolus of a game with non-empty imputation set contains exactly one element. Kopelowitz [10] proposed that the Nucleolus can be obtained by recursively solving a sequential linear programs (SLP):

$$LP_k : \quad \begin{array}{lll} \max \varepsilon & & \\ x(S) = v(S) + \varepsilon_r & \forall S \in \mathcal{J}_r & r = 0, 1, \cdots, k-1 \\ x(S) \geq v(S) + \varepsilon & \forall S \in 2^N \setminus \bigcup_{r=0}^{k-1} \mathcal{J}_r & \\ x \in \mathcal{I}(v). & & \end{array}$$

Here, $\mathcal{J}_0 = \{\emptyset, N\}$ and $\varepsilon_0 = 0$ initially; the number ε_r is the optimum value of the r-th program (LP$_r$), and $\mathcal{J}_r = \{S \in 2^N : x(S) = v(S) + \varepsilon_r \text{ for every } x \in X_r\}$, where $X_r = \{x \in \mathcal{I}(v) : (x, \varepsilon_r) \text{ is an optimal solution to } LP_r\}$. It can be shown that after at most $n - 1$ iterations, one arrives at a unique optimal solution (x^*, ε_k), where x^* is just the Nucleolus of the game. In addition, the set of optimal solutions X_1 to the first program LP$_1$ is called the least core of the game.

The definition of the Nucleolus entails comparisons between vectors of exponential length, and with linear programming approach, each linear programs in (SLP) may possess exponential size in the number of players. Clearly, both do not provide an efficient solution in general.

Flow games, first studied in Kailai and Zemel [8,9], arise from the profit distribution problem related to the maximum flow in a network. Let $D = (V, E; \omega; s, t)$ be a directed flow network, where V is the vertex set, E is the arc set, $\omega : E \to R^+$ is the arc capacity function, s and t are the source and the sink of the network, respectively. The network D is *simple* if $\omega(e) = 1$ for each $e \in E$, which is denoted briefly by $D = (V, E; s, t)$.

Definition 2 The flow game $\Gamma_f = (E, v)$ associated with network $D = (V, E; \omega; s, t)$ is defined by

(i) The player set is E;
(ii) $\forall S \subseteq E$, $v(S)$ is the value of a maximum flow from s to t in the subnetwork of D consisting only of arcs belonging to S.

Problem 1 (Computing the Nucleolus)
INSTANCE: A flow network $D = (V, E; \omega; s, t)$.
QUESTION: Is there a polynomial time algorithm to compute the Nucleolus of the flow game associated with D?

Problem 2 (Recognizing the Nucleolus)
INSTANCE: A flow network $D = (V, E; \omega; s, t)$ and $y : E \to R^+$.
QUESTION: Is it true that y is the Nucleolus of the flow game associated with D?

Key Results

Theorem 1 *Let $D = (V, E; s, t)$ be a simple network and $\Gamma_f = (E, v)$ be the associated flow game. Then the Nucleolus $\eta(v)$ can be computed in polynomial time.*

By making use of duality technique in linear programming, Kalai and Zemel [9] gave an characterization on the core of a flow game. They further conjectured that their approach may serve as a practical basis for computing the Nucleolus. In fact, the proof of Theorem 1 in the work of Deng Fang and Sun [2] is just an elegant application of Kalai and Zemel's approach (especially the duality technique), and hence settling their conjecture.

Theorem 2 *Given a flow game $\Gamma_f = (E, v)$ defined on network $D = (V, E; \omega; s, t)$, computing the Nucleolus $\eta(v)$ is $\mathcal{NP}$-hard.*

Theorem 3 *Given a flow game $\Gamma_f = (E, v)$ defined on network $D = (V, E; \omega; s, t)$, and an imputation $y \in \mathcal{I}(v)$, checking whether y is the Nucleolus of Γ_f is $\mathcal{NP}$-hard.*

Although a flow game can be formulated as a linear production game [1], the size of reduction may in general be exponential in space, and consequently, their complexity results on the Nucleolus are independent. However, in the $\mathcal{NP}$-hardness proof of Theorem 2 and 3, the flow game constructed possesses a polynomial size formulation of linear production game [2]. Therefore, as a direct corollary, the same $\mathcal{NP}$-hardness conclusions for linear production games are obtained. That is, both computing and recognizing the Nucleolus of a linear production game are $\mathcal{NP}$-hard.

Applications

As an important solution concept in economics and game theory, the Nucleolus and related solution concepts have been applied to study insurance policies, real estate and bankruptcy, etc. However, it is a challenging problem in mathematical programming to decide what classes of cooperative games permit polynomial time computation of the Nucleolus.

The first polynomial time algorithm for Nucleolus in a special tree game was proposed by Megiddo [11], in advocation of efficient algorithms for cooperative game solutions. Subsequently, some efficient algorithms have been developed for computing the Nucleolus, such as, for assignment games [13] and matching games [7]. On the negative side, $\mathcal{NP}$-hardness result was obtained for minimum cost spanning tree games [3].

Granot, Granot and Zhu [6] observed that most of the efficient algorithms for computing the Nucleolus are based on the fact that the information needed to completely characterize the Nucleolus is much less than that dictated by its definition. Therefore, they introduced the concept of a characterization set for the Nucleolus to embody the notion of "minimum" relevant information needed for determining the Nucleolus. Furthermore, based on the sequential linear programs (SLP), they established a general relationship between the size of a characterization set and the complexity of computing the Nucleolus. Following this line of development, some known efficient algorithms for computing the Nucleolus are derived directly.

Another approach to computing the Nucleolus is taken by Faigle, Kern and Kuipers [4], which is motivated by Schmeidler's observation that the Nucleolus of a game lies in the kernel [12]. In the case where the kernel of the game contains exactly one core vector and the minimum excess for any given allocation can be compute efficiently, their approach derives a polynomial time algorithm for the Nucleolus. This also generalizes some known results on the computation of the Nucleolus. However, their algorithm uses the ellipsoid method as a subroutine, it implies that the efficiency of the algorithm is of a more theoretical kind.

Open Problems

The field of combinatorial optimization has much to offer for the study of cooperative games. It is usually the case that the value of subgroup of players can be obtained via a combinatorial optimization problem, where the game is called a combinatorial optimization game. This class of games leads to the applications of a variety of combinatorial optimization techniques in design and analysis of algorithms, as well as establishing complexity results. One of the most interesting result is the LP duality characterization of the core [1]. However, little work dealt with the Nucleolus by using the duality technique so far. Hence, the work of Deng, Fang and Sun [2] on computing the Nucleolus may be of independent interest.

There are still many unsolved complexity questions concerning the Nucleolus. For the computation of the Nucleolus of matching games, Kern and Paulusma [7] proposed an efficient algorithm in unweighted case, and conjectured that it is in general $\mathcal{NP}$-hard. Since both simple flow game and matching game fall into the class of packing/covering games, it is interesting to know the complexity of computing the Nucleolus for other game models in this class, such as, vertex covering games and minimum coloring games.

For cooperative games arising from $\mathcal{NP}$-hard combinatorial optimization problems, the computation of the Nucleolus may in general be a hard task. For example, in a traveling salesman game, nodes of the graph are the players and an extra node 0, and the value of a subgroup S of players is the length of a minimum Hamiltonian tour in the subgraph induced by $S \cup \{0\}$ [1]. It would not be surprising if one shows that both the computation and the recognition of the Nucleolus for this game model are $\mathcal{NP}$-hard. However, this is not known yet. The same questions are proposed for facility location games [5], though there have been efficient algorithms for some special cases.

Moreover, when the computation of the Nucleolus is difficult, it is also interesting to seek for meaningful approximation concepts of the Nucleolus, especially from the political and economic background.

Cross References

► Complexity of Core
► Routing

Recommended Reading

1. Deng, X.: Combinatorial Optimization and Coalition Games. In: Du, D., Pardalos, P.M. (eds.) Handbook of combinatorial optimization, vol 2, pp 77–103, Kluwer, Boston (1998)
2. Deng, X., Fang, Q., Sun, X.: Finding Nucleolus of Flow Games. Proceedings of the 17th annual ACM-SIAM symposium on Discrete algorithm (SODA 2006). Lect. Notes in Comput. Sci. **3111**, 124–131 (2006)
3. Faigle, U., Kern, W., Kuipers, J.: Computing the Nucleolus of Min-cost Spanning Tree Games is $\mathcal{NP}$-hard. Int. J. Game Theory **27**, 443–450 (1998)
4. Faigle, U., Kern, W., Kuipers, J.: On the Computation of the Nucleolus of a Cooperative Game. Int. J. Game Theory **30**, 79–98 (2001)
5. Goemans, M.X., Skutella, M.: Cooperative Facility Location Games. J. Algorithms **50**, 194–214 (2004)
6. Granot, D., Granot, F., Zhu, W.R.: Characterization Sets for the Nucleolus. Int. J. Game Theory **27**, 359–374 (1998)
7. Kern, W., Paulusma, D.: Matching Games: The Least Core and the Nucleolus. Math. Oper. Res. **28**, 294–308 (2003)
8. Kalai, E., Zemel, E.: Totally Balanced Games and Games of Flow. Math. Oper. Res. **7**, 476–478 (1982)
9. Kalai, E., Zemel, E.: Generalized Network Problems Yielding Totally Balanced Games. Oper. Res. **30**, 998–1008 (1982)

10. Kopelowitz, A.: Computation of the Kernels of Simple Games and the Nucleolus of *n*-person Games. RM-31, Math. Dept., The Hebre University of Jerusalem (1967)
11. Megiddo, N.: Computational Complexity and the Game Theory Approach to Cost Allocation for a Tree. Math. Oper. Res. **3**, 189–196 (1978)
12. Schmeidler, D.: The Nucleolus of a Characteristic Function Game. SIAM J. Appl. Math. **17**, 1163–1170 (1969)
13. Solymosi, T., Raghavan, T.E.S.: An Algorithm for Finding the Nucleolus of Assignment Games. Int. J. Game Theory **23**, 119–143 (1994)

O

Oblivious Routing

2002; Räcke

NIKHIL BANSAL
IBM Research, IBM, Yorktown Heights, NY, USA

Keywords and Synonyms

Fixed path routing

Problem Definition

Consider a communication network, for example, the network of cities across the country connected by communication links. There are several sender-receiver pairs on this network that wish to communicate by sending traffic across the network. The problem deals with routing all the traffic across the network such that no link in the network is overly congested. That is, no link in the network should carry too much traffic relative to its capacity. The obliviousness refers to the requirement that the routes in the network must be designed without the knowledge of the actual traffic demands that arise in the network, i. e. the route for every sender-receiver pair stays fixed irrespective of how much traffic any pair chooses to send. Designing a good oblivious routing strategy is useful since it ensures that the network is robust to changes in the traffic pattern.

Notations

Let $G = (V, E)$ be an undirected graph with non-negative capacities $c(e)$ on edges $e \in E$. Suppose there are k source-destination pairs (s_i, t_i) for $i = 1, \dots, k$ and let d_i denote the amount of flow (or demand) that pair i wishes to send from s_i to t_i. Given a routing of these flows on G, the congestion of an edge e is defined as $u(e)/c(e)$, the ratio of the total flow crossing edge e divided by its capacity. The congestion of the overall routing is defined as the maximum congestion over all edges. The congestion minimization problem is to find the routing that minimizes the maximum congestion. Observe that specifying a flow from s_i to t_i is equivalent to finding a probability distribution (not necessarily unique) on a collection of paths from s_i to t_i.

The congestion minimization problem can be studied in many settings. In the offline setting, the instance of the flow problem is provided in advance and the goal is to find the optimum routing. In the on line setting, the demands arrive in an arbitrary adversarial order and a flow must be specified for a demand immediately upon arrival; this flow is fixed forever and cannot be rerouted later when new demands arrive. Several distributed approaches have also been studied where each pair routes its flow in a distributed manner based on some global information such as the current congestion on the edges.

In this note, the oblivious setting is considered. Here a routing scheme is specified for each pair of vertices in advance without any knowledge of which demands will actually arrive. Note that an algorithm in the oblivious setting is severely restricted. In particular, if d_i units of demand arrive for pair (s_i, t_i), the algorithm must necessarily route this demand according to the pre-specified paths irrespective of the other demands or any other information such as congestion of other edges. Thus given a network graph G, the oblivious flows need to be computed just once. After this is done, the job of the routing algorithm is trivial; whenever a demand arrives, it simply routes it along the pre-computed path. An oblivious routing scheme is called c-competitive if for any collection of demands D, the maximum congestion of the oblivious routing is no more than c times the congestion of the optimum offline solution for D. Given this stringent requirement on the quality of oblivious routing, it is not a priori clear that any reasonable oblivious routing scheme should exist at all.

Key Results

Oblivious routing was first studied in the context of permutation routing where the demand pairs form a permutation and have unit value each. It was shown that any oblivious routing that specifies a single path (instead of a flow) between every two vertices must necessarily perform badly. This was first shown by Borodin and Hopcroft [6] for hypercubes and the argument was later extended to general graphs by Kaklamanis, Krizanc and Tsantilas [12], who showed the following.

Theorem 1 ([6,12]) *For every graph G of size n and maximum degree d and every oblivious routing strategy using only a single path for every source-destination pair, there is a permutation that causes an overlap of at least* $(n/d)^{1/2}$ *paths at some node. Thus if each edge in G has unit capacity, the edge congestion is at least* $(n/d)^{1/2}/d$.

Since there exists constant degree graphs such as the butterfly graphs that can route any permutation with logarithmic congestion, this implies that such oblivious routing schemes must necessarily perform poorly on certain graphs.

Fortunately, the situation is substantially better if the single path requirement is relaxed and a probability distribution on paths (equivalently a flow) is allowed between each pair of vertices. In a seminal paper, Valiant and Brebner [17] gave the first oblivious permutation routing scheme with low congestion on the hypercube. It is instructive to consider their scheme. Consider an hypercube with $N = 2^n$ vertices. Represent vertex i by the binary expansion of i. For any two vertices s and t, there is a canonical path (of length at most $n = \log N$) from s to t obtained by starting from s and flipping the bits of s in left to right order to match with that of t. Consider routing scheme that for a pair s and t, it first chooses some node p uniformly at random, routes the flow from s to p along the canonical path, and then routes it again from p to t along the canonical path (or equivalently it sends $1/N$ units of flow from s to each intermediate vertex p and then routes it to t). An relatively simple analysis shows that

Theorem 2 ([17]) *The above oblivious routing scheme achieves a congestion of* $O(1)$ *for hypercubes.*

Subsequently, oblivious routing schemes were proposed for few other special classes of networks. However, the problem of designing oblivious routing schemes for general graphs remained open until recently, when in a breakthrough result Räcke showed the following.

Theorem 3 ([15]) *For any undirected capacitated graph* $G = (V, E)$, *there exist an oblivious routing scheme with congestion* $O(\log^3 n)$ *where n is the number of vertices in G.*

The key to Räcke's theorem is a hierarchical decomposition procedure of the underlying graph (described in further detail below). This hierarchical decomposition is a fundamental combinatorial result about the cut structure of graphs and has found several other applications, some of which are mentioned in Section "Applications". Räcke's proof of Theorem 3 only showed the existence of a good hierarchical decomposition and did not give an efficient polynomial time algorithm to find it. In subsequent work, Harrelson, Hildrum and Rao [11] gave a polynomial time procedure to find the decomposition and improved the competitive ratio of the oblivious routing to $O(\log^2 n \log\log n)$.

Theorem 4 ([11]) *There exists an* $O(\log^2 n \log\log n)$*-competitive oblivious routing scheme for general graphs and moreover it can be found in polynomial time.*

Interestingly, Azar et al. [4] show that the problem of finding the optimum oblivious routing for a graph can be formulated as a linear program. They consider a formulation with exponentially many constraints; one for each possible demand matrix that has optimum congestion 1, that enforces that the oblivious routing should have low congestion for this demand matrix. Azar et al. [4] give a separation oracle for this problem and hence it be solved using the ellipsoid method. A more practical polynomial size linear program was given later by Applegate and Cohen [2]. Bansal et al. [5] considered a more general variant referred to as the online oblivious routing that can also be used to find an optimum oblivious routing. However, note that without Räcke's result, it would not be clear whether these optimum routings were any good. Moreover these techniques do not give a hierarchical decomposition, and hence may be less desirable in certain contexts. On the other hand, they may be more useful sometimes since they produce an optimum routing (while [11] implies an $O(\log^2 n \log\log n)$-competitive routing for any graph, the best oblivious routing could have a much better guarantee for a specific graph).

Oblivious routing has also been studied for directed graphs, however the situation is much worse here. Azar et al. [4] show that there exist directed graphs where any oblivious routing is $\Omega(\sqrt{n})$ competitive. Some positive results are also known [10]. Hajiaghayi et al. [8] show a substantially improved guarantee of $O(\log^2 n)$ for directed graphs in the random demands model. Here each source-sink pair has a distribution (that is known by the algorithm) from which it chooses its demand independently. A relaxation of oblivious routing known as semi-oblivious routing has also been studied recently [9].

Techniques

This section describes the high level idea of Räcke's result. For a subset $S \subset V$, let $\text{cap}(S)$ denote the total capacity of the edges that cross the cut $(S, V \setminus S)$ and let $\text{dem}(S)$

denote the total demand that must be routed across the cut $(S, V \setminus S)$. Observe that $q = \max_{S \subset V} \text{dem}(S)/\text{cap}(S)$ is a lower bound on the congestion of any solution. On the other hand, the key result [3,13] relating multi-commodity flows and cuts implies that there is a routing such that the maximum congestion is at most $O(q \log k)$ where k is the number of distinct source sink pairs. However, note that this by itself does not suffice to obtain good oblivious routings, since a pair (s_i, t_i) can have different routing for different demand sets. The main idea of Räcke was to impose a tree like structure for routing on the graph to achieve obliviousness. This is formalized by a hierarchical decomposition described below.

Consider a hierarchical decomposition of the graph $G = (V, E)$ as follows. Starting from the set $S = V$, the sets are partitioned successively until each set becomes singleton vertex. This hierarchical decomposition can be viewed naturally as a tree T, where the root corresponds to the set V, and leaves corresponds to the singleton sets $\{v\}$. Let S_i denote the subset of V corresponding to node i in T. For an edge (i, j) in the tree where i is the child of j, assign it a capacity equal to $\text{cap}(S_i)$ (note that this is the capacity from S_i to the rest of G and not just capacity between S_i and S_j in G). The tree T is used to simulate routing in G and vice versa. Given a demand from u to v in G, consider the corresponding (unique) route among leaves corresponding to $\{u\}$ and $\{v\}$ in T. For any set of demands, it is easily seen that the congestion in T is no more than the congestion in G. Conversely, Räcke showed that there also exists a tree T where the routes in T can be mapped back to flows in G, such that for any set of demands the congestion in G is at most $O(\log^3 n)$ times that in T. In this mapping a flow along the (i, j) in the tree T corresponds to a suitably constructed flow between sets S_i and S_j in G. Since route between any two vertices in T is unique, this gives an oblivious routing in G.

Räcke uses very clever ideas to show the existence of such a hierarchical decomposition. Describing the construction is beyond the scope of this note, but it is instructive to understand the properties that must be satisfied by such a decomposition. First, the tree T should capture the bottlenecks in G, i. e. if there is a set of demands that produces high congestion in G, then it should also produce a high congestion in T. A natural approach to construct T would be to start with V, split V along a bottleneck (formally, along a cut with low sparsity), and recurse. However, this approach is too simple to work. As discussed below, T must also satisfy two other natural conditions, known as the *bandwidth* property and the *weight* property which are motivated as follows. Consider a node i connected to its parent j in T. Then, i needs to route $\text{dem}(S_i)$ flow out of S_i and it incurs congestion $\text{dem}(S_i)/\text{cap}(S_i)$ in T. However, when T is mapped back to G, all the flow going out of S_i must pass via S_j. To ensure that the edges from S_i to S_j are not overloaded, it must be the case that the capacity from S_i to S_j is not too small compared to the capacity from S_i to the rest of the graph $V \setminus S_i$. This is referred to as the bandwidth property. Räcke guarantees that this is ratio is always $\Omega(1/\log n)$ for every S_i and S_j corresponding to edges (i, j) in the tree. The weight property is motivated as follows. Consider a node j in T with children $i_1, \dots, i_p$, then the weight property essentially requires that the sets $S_{i_1}, \dots, S_{i_p}$ should be well connected among themselves even when restricted to the subgraph S_j. To see why this is needed, consider any communication between, say nodes i_1 and i_2 in T. It takes the route i_1 to j to i_2, and hence in G, S_{i_1} cannot use edges that lie outside S_j to communicate with S_{i_2}. Räcke shows that these conditions suffice and that a decomposition can be obtained that satisfies them.

The factor $O(\log^3 n)$ in Räcke's guarantee arises from three sources. The first logarithmic factor is due to the flow-cut gap [3,13]. The second is due to the logarithmic height of the tree, and the third is due to the loss of a logarithmic factor in the bandwidth and weight properties.

Applications

The problem has widespread applications to routing in networks. In practice it is often required that the routes must be a single path (instead of flows). This can often be achieved by randomized rounding techniques (sometimes under an assumption that the demands to capacity ratios be not too large). The flow formulation provides a much cleaner framework for studying the problems above.

Interestingly, the hierarchical decomposition also found widespread uses in other seemingly unrelated areas such as obtaining good pre-conditioners for solving systems of linear equations [14,16], for the all-or-nothing multicommodity flow problem [7], online network optimization [1] and so on.

Open Problems

It is possible that any graph has an $O(\log n)$ competitive oblivious routing scheme. Settling this is a key open question.

Cross References

- ► Routing
- ► Separators in Graphs
- ► Sparsest Cut

Recommended Reading

1. Alon, N., Awerbuch, B., Azar, Y., Buchbinder, N., Naor, J.: A general approach to online network optimization problems. In: Symposium on Discrete Algorithms, pp. 570–579 (2004)
2. Applegate, D., Cohen, E.: Making intra-domain routing robust to changing and uncertain traffic demands: understanding fundamental tradeoffs. In: SIGCOMM, pp. 313–324 (2003)
3. Aumann, Y., Rabani, Y.: An O(log k) approximate min-cut max-flow theorem and approximation algorithm. SIAM J. Comput. **27**(1), 291–301 (1998)
4. Azar, Y., Cohen, E., Fiat, A., Kaplan, H., Räcke, H.: Optimal oblivious routing in polynomial time. In: Proceedings of the 35th ACM Symposium on the Theory of Computing, pp. 383–388 (2003)
5. Bansal, N., Blum, A., Chawla, S., Meyerson, A.: Online oblivious routing. In Symposium on Parallelism in Algorithms and Architectures, pp. 44–49 (2003)
6. Borodin, A., Hopcroft, J.: Routing, merging and sorting on parallel models of computation. J. Comput. Syst. Sci. **10**(1), 130–145 (1985)
7. Chekuri, C., Khanna, S., Shepherd, F.B.: The All-or-Nothing Multicommodity Flow Problem. In: Proceedings of 36th ACM Symposium on Theory of Computing, pp. 156–165 (2004)
8. Hajiaghayi, M., Kim, J.H., Leighton, T., Räcke, H.: Oblivious routing in directed graphs with random demands. In: Symposium on Theory of Computing, pp. 193–201 (2005)
9. Hajiaghayi, M., Kleinberg, R., Leighton, T.: Semi-oblivious routing: lower bounds. In: Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms, pp. 929–938 (2007)
10. Hajiaghayi, M., Kleinberg, R., Leighton, T., Räcke, H.: Oblivious routing in node-capacitated and directed graphs. In: Proceedings of the 16th ACM-SIAM Symposium on Discrete Algorithms, pp. 782–790 (2005)
11. Harrelson, C., Hildrum, K., Rao, S.: A polynomial-time tree decomposition to minimize congestion. In: Proceedings of the 15th annual ACM Symposium on Parallel Algorithms and Architectures, pp. 34–43 (2003)
12. Kaklamanis, C., Krizanc, D., Tsantilas, T.: Tight bounds for oblivious routing in the hypercube. In: Proceedings of the 3rd annual ACM Symposium on Parallel Algorithms and Architectures, pp. 31–36 (1991)
13. Linial, N., London, E., Rabinovich, Y.: The geometry of graphs and some of its algorithmic applications. In: IEEE Symposium on Foundations of Computer Science, pp. 577–591 (1994)
14. Maggs, B.M., Miller, G.L., Parekh, O., Ravi, R., Woo, S.L.M.: Finding effective support-tree preconditioners. In: Symposium on Parallel Algorithms and Architectures, pp. 176–185 (2005)
15. Räcke, H.: Minimizing congestion in general networks. In: Proceedings of the 43rd Annual Symposium on the Foundations of Computer Science, pp. 43–52 (2002)
16. Vaidya, P.: Solving linear equations with symmetric diagonally dominant matrices by constructing good preconditioners. Unpublished manuscript (1991)
17. Valiant, L., Brebner, G.J.: Universal schemes for parallel communication. In: Proceedings of the 13th ACM Symposium on Theory of Computing, pp. 263–277 (1981)

Obstacle Avoidance Algorithms in Wireless Sensor Networks

2007; Powell, Nikoletseas

SOTIRIS NIKOLETSEAS[1], OLIVIER POWELL[2]
[1] Computer Engineering and Informatics Department, Computer Technology Institute, Patras, Greece
[2] Informatics Department, University of Geneva, Geneva, Switzerland

Keywords and Synonyms

Greedy geographic routing; Routing holes

Problem Definition

Wireless sensor networks are composed of many small devices called sensor nodes with sensing, computing and radio frequency communication capabilities. Sensor nodes are typically deployed in an ad hoc manner and use their sensors to collect environmental data. The emerging network collectively processes, aggregates and propagates data to regions of interest, e. g. from a region where an event is being detected to a base station or a mobile user. This entry is concerned with the data propagation duty of the sensor network in the presence of obstacles.

For different reasons, including energy conservation and limited transmission range of sensor nodes, information propagation is achieved via multi-hop message transmission, as opposed to single-hop long range transmission. As a consequence, message routing becomes necessary. Routing algorithms are usually situated at the network layer of the protocol stack where the most important component is the (dynamic) communication graph.

Definition 1 (Communication graph) A wireless sensor network is viewed as a graph $G = (V, E)$ where vertexes correspond to sensor nodes and edges represent wireless links between nodes.

Wireless sensor networks have stringent constraints that make classical routing algorithms inefficient, unreliable or even incorrect. Therefore, the specific requirements of wireless sensor networks have to be addressed [2] and geographic routing offers the possibility to design particularly well adapted algorithms.

Geographic Routing

A geographic routing algorithm takes advantage of the fact that sensor nodes are location aware, i. e. they know their position in a coordinate system following the use of a localization protocol [7]. Although likely to introduce a significant overhead, the use of a localization protocol is also likely to be inevitable in many applications where environmental data collected by the sensors would be useless if not related to some geographical information. For those applications, node location awareness can be assumed to be available for routing purposes at no additional cost.

The Power of Simple Geographic Routing The early "most forward within range" (MFR) or greedy geographic routing algorithms [14] route messages by maximizing, at each hop, the progress on a projected line towards the destination or, alternatively, minimizing the remaining distance to the message's destination. Both of these greedy heuristics are referred to as *greedy forwarding* (GF). Greedy forwarding is a very appealing routing technique for wireless sensor networks. Among explanations for the attractiveness of GF are the following. (1) GF, as is almost imperatively required, is *fully distributed*. (2) It is *lightweight* in the sense that it induces no topology control overhead. (3) It is *all-to-all* (as opposed to all-to-one). (4) Making no assumptions on the structure of the communication graph, which can be directed, undirected, stable or dynamic (e. g. nodes may be mobile or wireless links may appear and disappear, for example following environmental fluctuation or as a consequence of lower protocol stack layers such as sleep/awake schemes for energy saving purposes), it is *robust*. (5) It is *on-demand*: no routing table or gradient has to be built prior to message propagation. (6) *Efficiency* is featured as messages find short paths to their destination in terms of hop count. (7) It is very *simple* and thus *easy to implement*. (8) It is *memory efficient* in the sense that (8a) the only information stored in the message header is the message's destination and that (8b) it is "ecologically sound" because no "polluting" information is stored on the sensor nodes visited by messages.

Problem Statement

Although very appealing, GF suffers from a major flaw: when a message reaches a local minimum where no further progress towards the destination is possible the routing algorithm fails. There are two major reasons for the occurrence of local minimums: routing holes [1] and obstacles.

Definition 2 The so called *routing holes* are low density regions of the network where no sensor nodes are available for next-hop forwarding.

Even in uniform-randomly deployed networks, routing holes appear as the manifestation of statistical variance of node density. Although increasing as network density diminishes, routing holes have a severe impact on the performance of GF even for very high density networks [12].

Definition 3 A *transmission blocking obstacle* is a region of the network where no sensors are deployed and through which radio signals do not propagate.

Clearly, large obstacles lying between a message and its destination tend to make GF fail.

The problem reported in this entry is to find a geographic routing algorithm that *maintains the advantages of greedy forwarding* listed in Sect. "Geographic Routing" such as simplicity, light weight, robustness and efficiency while *overcoming its weaknesses*: the inability to escape local minimum nodes created by routing holes and large transmission blocking obstacles such as those seen in Fig. 1.

Problem 1 (Escaping routing holes) *The first problem is to route messages out of the many routing holes which are statistically doomed to occur even in dense networks.*

Problem 2 (Contouring obstacles) *The second problem is to design a protocol capable of routing messages around large transmission blocking obstacles.*

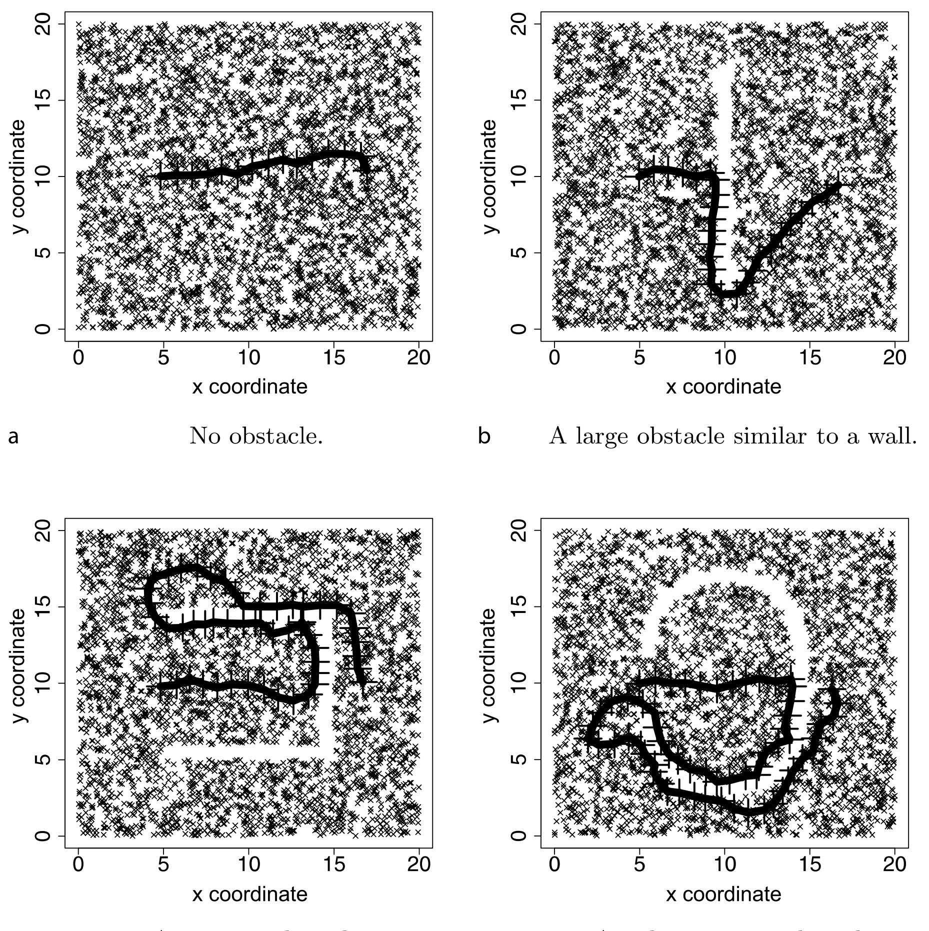
Problem 1 can be considered a simplified instance of problem 2. Lightweight solutions to problem 1 have been previously proposed, usually using limited backtracking [6] or controlled flooding combined with a GF heuristic [4,13]. However, as shown in [5] where an integrated model for obstacles is proposed and where different algorithms are compared with respect to their obstacle avoidance capability, those solutions do not satisfactorily solve problem 2 in the sense that only small and simple obstacles are efficiently bypassed.

Key Results

In [12] a new **g**eographic **r**out**i**ng around obsta**c**les (GRIC) algorithm was proposed to address the problems described in the previous section.

Basic Idea of the Algorithm

In GF, the strategy is to always propagate the message to the neighbor that maximizes progress *towards the destination*. Similarly, GRIC also maximizes progress in a chosen direction. However, this direction is not necessarily the message's destination but an *ideal direction of progress* which has to be computed according to one of two possible strategies: the *inertia mode* or the *rescue mode* described below. Finally, it was found that performance is better in the presence of slightly unstable networks, c. f.



a No obstacle.

b A large obstacle similar to a wall.

c A concave obstacle

d Another concave obstacle

Obstacle Avoidance Algorithms in Wireless Sensor Networks, Figure 1
Typical path followed by GRIC to bypass certain obstacles

Result 4, and thus in the case where the communication graph is very stable it is recommended to use a randomized version of GRIC where nodes about to take a routing decision randomly mark as either passive or active each outbound wireless link of the communication graph. Only active wireless links can be used for message propagation, and link status is re-evaluated each time a new routing decision is taken. Marking links as active with a probability of $p = 0.95$ was found to be a good choice for practical purposes [12].

Inertia Mode The idea of the inertia mode is that a message should have a strong incentive to go towards its destination but this incentive should be moderated by one to follow the straight ahead direction of current motion "*... like a celestial body in a planet system ...*" [12]. The inertia mode aims at making messages follow closely the perimeter of routing holes and obstacles in order to eventually bypass them and ensure final routing to the destination. To implement the inertia mode, a single additional assumption is made: sensor nodes should be aware of the position of the node from which they receive a message. As an example, this could be done by piggy-backing this 1-hop away routing path history in the message header. Knowing its own position p, the message's destination and the 1-hop away previous position of the message a sensor node can compute the vectors v_{cur} and v_{dst} starting at position p and pointing in the direction of current motion and the direction to the message's destination respectively. The inertia mode defines the ideal direction of progress, v_{idl}, as a vector starting at point p and lying "somewhere in between" v_{cur} and v_{dst}. More precisely, let α be the only angle in $[-\pi, \pi[$ such that v_{dst} is obtained by applying a rotation of angle α to v_{cur}, then v_{idl} is the vector obtained by applying a rotation of angle α' to v_{cur}, where $\alpha' = \mathrm{sign}(\alpha) \cdot \min\left\{\frac{\pi}{6}, |\alpha|\right\}$. Finally, the message is greedily forwarded to the neighbor node maximizing progress in the computed ideal direction of progress v_{idl}.

Rescue Mode In order to improve overall performance and to bypass complex obstacles, the rescue mode imitates the right-hand rule (RHR) which is a well known wall follower technique to find one's way out of a maze. A high-level description of the RHR component of GRIC is given below while details will be found in [12]. In GRIC, the RHR makes use of a virtual compass and a flag. The virtual compass assigns to v_{cur} a cardinal point value, treating the message's destination as the north. Considering the angle α defined in the previous section, the compass returns a string x-y with x equal to north or south if $|\alpha|$ is smaller or greater than $\frac{\pi}{2}$ respectively, while y is equal to west or east if α is negative or positive respectively. The first time the compass returns a south value, the flag is raised and tagged with the (x, y) value of the compass. Raising the flag means that the message is being routed around an obstacle using the RHR rule if the compass indicates south-west. In the case where the compass indicates south-east, a symmetric case not discussed here for brevity is applied using the left-hand rule (LHR) instead of the RHR. Once the flag is raised, it stays up with its tag unchanged until the compass indicates north, meaning that the obstacle has been bypassed. In fact, a small optimization can be made by lowering the flag only if the compass points to the north-west (in the case of the RHR) and not if it points north-east, but c.f. [12] for details. According to the RHR the obstacle's perimeter should be followed closely and kept on the right side of the message's current direction. If ever the compass and the flag's tag disagree, i. e. if the flag is tagged with south-west and the compass returns south-east, it is assumed that the message is turning left too much, that it risks going away from the obstacle and that the RHR is at risk of being violated (a symmetric case applies for the LHR). When this is so, GRIC responds by calling the rescue mode which changes the default way of computing v_{idl}: in rescue mode the message is forced to turn right (or left if the LHR is applied), by defining v_{idl} as the vector obtained by applying to v_{cur} a rotation of angle α'' (instead of α' in inertia mode) where $\alpha'' = -sign(\alpha)(2\pi - |\alpha|)/6$.

Main Findings

The performance of GRIC was evaluated through simulations. The main parameters were the presence (or absence) of different shapes of large communication blocking obstacles and the network density which ranged from very low to very high and controls the average degree of the communication graph and the occurrence of routing holes. The main performance metrics were the success rate, i. e. the percentage of messages routed to destination, and the path length. The main findings are that GRIC efficiently, i. e. using short paths, bypasses routing holes and obstacles but that in the presence of hard obstacles, the performance decreases with network density. In Figure 1, typical routing paths found by GRIC for different obstacle shapes are illustrated, c. f. [12] for details on the simulation environment.

Result 1 *In the absence of obstacles, routing holes are bypassed for every network density: The success rate is close to 100% as long as the source and the destination are connected. Also, routing is efficient in the sense that path lengths are very short.*

Result 2 *Some convex obstacles such as the one in Fig. 1b are bypassed with almost 100% success rate and using short paths, even for low densities. When the density gets very low performance diminishes: If the density gets below the critical level guaranteeing the communication graph to be connected with high probability, then the success probability diminishes quickly and successful routings use longer routing paths.*

Result 3 *Some large concave obstacles such as those in Fig. 1c and d are efficiently bypassed. However, when facing such obstacles performance becomes more sensitive to network density. The success rate drops and routing paths become longer when the density gets below a certain level depending on the exact obstacle shape.*

Result 4 (Robustness) *Similarly to* GF, GRIC *is robust to link instability. Furthermore, it was observed that limited link instability has a significantly positive impact on performances. This can be understood as the fact that messages are less likely to enter endless routing loops in a "hot" system than in a "cold" system.*

Applications

Replacement for Greedy Forwarding

Because it makes no compromise with the advantages of GF except the fact that it may be somehow more complicated to implement and because it overcomes GF's main limitations, GRIC can probably replace GF for most routing scenarios including but not exclusively wireless sensor networks. As an example opportunistic-routing strategies [11] could be applied to GRIC rather than to GF.

Wireless Sensor Networks with Large Obstacles

GRIC successfully bypasses large communication blocking obstacles. However, it does so efficiently only if the network density is high enough. This suggests that the obstacle avoidance feature of GRIC may be more useful for dense wireless networks than for sparse networks. Wireless sensor networks are an example of networks which are usually considered to be dense.

Dynamic Networks

There exist some powerful alternatives to GRIC such as the celebrated guaranteed delivery protocols GFG [3], GPSR [8] or GOAFR [10]. Those protocols rely on a planarization phase such as the lazy cross-link detection protocol (CLDP) [9]. LCR implies significant topology maintenance overhead which would be amortized over time if the network is stable enough. On the contrary, if the network is highly dynamic the necessity for frequent updates could make this topology maintenance overhead prohibitive. GRIC may thus be a preferable choice for dynamic networks where the communication graph is not a stable structure.

Open Problems

(1) Hard concave obstacles such as the one in Figure 1d are still a challenge for lightweight protocol since in this configuration GRIC's performance is strongly dependent on network density. (2) Low to very low densities are challenging when combined with large obstacles, even when they are "simple" convex obstacles like the one in Figure 1b. (3) The problem reported in this entry in the case of 3-dimensional networks is open. Inertia may be of some help, however the virtual compass and the right-hand rule seem quite strongly depend-ant on the 2-dimensional plane. (4) GRIC is not loop free. A mechanism to detect loops or excessively long routing paths would be quite important for practical purposes. (5) The understanding of GRIC could be improved. Analytical results are lacking and new metrics could be considered such as network lifetime, energy consumption or traffic congestion.

Cross References

► Probabilistic Data Forwarding in Wireless Sensor Networks

Recommended Reading

1. Ahmed, N., Kanhere, S.S., Jha, S.: The holes problem in wireless sensor networks: a survey. SIGMOBILE Mob. Comput. Commun. Rev. **9**, 4–18 (2005)
2. Al-Karaki, J.N., Kamal, A.E.: Routing techniques in wireless sensor networks: a survey. Wirel. Commun. IEEE **11**, 6–28 (2004)
3. Bose, P., Morin, P., Stojmenovic, I., Urrutia, J.: Routing with guaranteed delivery in ad hoc wireless networks. In: Discrete Algorithms and Methods for Mobile Computing and Communications (1999)
4. Chatzigiannakis, I., Dimitriou, T., Nikoletseas, S., Spirakis, P.: A probabilistic forwarding protocol for efficient data propagation in sensor networks. In: European Wireless Conference on Mobility and Wireless Systems beyond 3G (EW 2004), pp. 344–350. Barcelona, Spain, 27 February 2004
5. Chatzigiannakis, I., Mylonas, G., Nikoletseas, S.: Modeling and evaluation of the effect of obstacles on the performance of wireless sensor networks. In: 39th ACM/IEEE Simulation Symposium (ANSS), Los Alamitos, CA, USA, IEEE Computer Society, pp. 50–60 (2006)
6. Chatzigiannakis, I., Nikoletseas S., Spirakis, P.: Smart dust protocols for local detection and propagation. J. Mob. Netw. (MONET) **10**, 621–635 (2005)
7. Karl, H., Willig, A.: Protocols and Architectures for Wireless Sensor Networks. Wiley, West Sussex (2005)
8. Karp, B., Kung, H.T.: GPSR: greedy perimeter stateless routing for wireless networks. In: Mobile Computing and Networking. ACM, New York (2000)
9. Kim, Y.J., Govindan, R., Karp, B., Shenker, S.: Lazy cross-link removal for geographic routing. In: Embedded Networked Sensor Systems. ACM, New York (2006)
10. Kuhn, F., Wattenhofer, R., Zhang, Y., Zollinger, A.: Geometric ad-hoc routing: of theory and practice. In: Principles of Distributed Computing. ACM, New York (2003)
11. Lee, S., Bhattacharjee, B., Banerjee, S.: Efficient geographic routing in multihop wireless networks. In MobiHoc '05: Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing, pp. 230–241. ACM, New York (2005)
12. Powell, O., Nikolesteas, S.: Simple and efficient geographic routing around obstacles for wireless sensor networks. In: WEA 6th Workshop on Experimental Algorithms, Rome, Italy. Springer, Berlin (2007)
13. Stojmenovic, I., Lin, X.: Loop-free hybrid single-path/flooding routing algorithms with guaranteed delivery for wireless networks. IEEE Trans. Paral. Distrib. Syst. **12**, 1023–1032 (2001)
14. Takagi, H., Kleinrock, L.: Optimal transmission ranges for randomly distributed packet radio terminals. Communications, IEEE Trans. [legacy, pre - 1988]. **32**, 246–257 (1984)

O(log log *n*)-competitive Binary Search Tree

2004; Demaine, Harmon, Iacono, Patrascu

CHENGWEN CHRIS WANG
Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA

Keywords and Synonyms

Tango

Problem Definition

Here is a precise definition of BST algorithms and their costs. This model is implied by most BST papers, and developed in detail by Wilber [22]. A static set of n keys is stored in the nodes of a binary tree. The keys are from a totally ordered universe, and they are stored in symmetric order. Each node has a pointer to its left child, to its right child, and to its parent. Also, each node may keep $o(\log n)$ bits of additional information but no additional pointers.

A BST algorithm is required to process a sequence of m accesses (without insertions or deletions), $S = s_1, s_2, s_3, s_4 \ldots s_m$. The ith access starts from the root and follows pointers until s_i is reached. The algorithm can update the fields in any node or rotate any edges that it touches along the way. The cost of the algorithm to execute an access sequence is defined to be the number of nodes touched plus the number of rotations.

Let A be any BST algorithm, define $A(S)$ as the cost of performing sequence S and $\mathrm{OPT}(S, T_0)$ as the minimum cost to perform the sequence S. An algorithm A is T-competitive if for all possible sequences S, $A(S) \le T * \mathrm{OPT}(S, T_0) + O(m + n)$.

Since the number of rotation needed to change any binary tree of n keys into another one (with the same n keys) is at most $2n - 6$ [4,5,12,13,15]. It follows that $\mathrm{OPT}(S, T_0)$ differs from $\mathrm{OPT}(S, T'_0)$ by at most $2n - 6$. Thus, if $m > n$, then the initial tree can only affect the constant factor.

Key Results

The *interleave bound* is a lower bound on $\mathrm{OPT}(S, T_0)$ that depends only on S. Consider any binary search tree P of all the elements in T_0. For each node y in P, define the *left side* of y includes all nodes in y's left subtree and y. And define the *right side* of y includes all nodes in y's right subtree. For each node y, label each access s_i in S by whether it is in the left or right side of y, ignoring all accesses not in y's subtree. Denote the number of times the label changes for y as $\mathrm{IB}(S, y)$. The *interleave* bound $\mathrm{IB}(S)$ is $\sum_y \mathrm{IB}(S, y)$.

Theorem 1 (Interleave Lower Bound [6,22]) $\mathrm{IB}(S)/2 - n$ *is a lower bound on* $\mathrm{OPT}(S, T_0)$.

Demaine et al observes that it is impossible to use this lower bound to improve the competitive ratio beyond $\Theta(\log \log n)$.

Theorem 2 (Tango is $O(\log \log n)$-competitive BST [6]) *The running time of Tango BST on a sequence S of m accesses is* $O((\mathrm{OPT}(S, T_0)) + n) * (1 + \log \log n))$.

Applications

Binary search tree (BST) is one of the oldest data structures in the history of computer science. It is frequently used to maintain an ordered set of data. In the last 40 years, many specialized binary search trees have been designed for specific applications. Almost every one of them supports access, insertion and deletion in worst-case $O(\log n)$ time on average for random sequences of access. This matches the best theoretically possible worst-case bound. For most of these data structures, a random sequence of m accesses will use $\Theta(m \log n)$ time.

While it is impossible to have better asymptotic performance for a random sequence of m accesses, many of the real world access sequences are not random. For instance, if the set of accesses are randomly drawn from a *small* subset of k element, it's possible to answer all the accesses in $O(m \log k)$ time. A notable binary search tree is Splay Tree. It is proved to perform well for many access patterns [2,3,8,14,16,17,18]. As a result, Sleator and Tarjan [14] conjectured that splay tree is $O(1)$-competitive to the optimal off-line BST. After more than 20 years, the conjecture remains an open problem.

Over the years, several restricted types of optimality have been proved. Many of these restrictions and usage patterns are based on real world applications. If each access is drawn independently at random from a fixed distribution, D, Knuth [11] constructed a BST based on D that is expected to run in optimal time up to a constant factor. Sleator and Tarjan [14] achieve the same bound without knowing D ahead of time. Other types includes key-independent optimality [10] and BST with free rotations [1].

In 2004, Demaine et al suggested searching for alternative BST algorithms that have small but non-constant competitive factors [6]. They proposed *Tango*, the first data structure proved to achieve a non-trivial competitive factor of $O(\log \log n)$. This is a major step toward developing a $O(1)$-competitive BST, and this line of research could potentially replace a large number of specialized BSTs.

Open Problems

Following this paper, several new $O(\log \log n)$-competitive BST have emerged [9,21]. A notable example is Multi–Splay Trees [21]. It generalizes the interleave bound to include insertions and deletions. Multi–Splay Trees also have many theorems analogous to Splay Trees [20,21], such as the access lemma and the working set theorem. Wang [21] conjectured that Multi-Splay Trees is $O(1)$-competitive, but it remains an open problem.

Returning to the original motivation for this research, the problem of finding an $o(\log \log n)$-competitive on-line

BST remains open. Several attempts have been made to improve the lower bound [6,7,22], but none of them have led to a lower competitive ratio. Even in the off-line model, the problem of finding an $O(1)$-competitive BST is difficult. The best known off-line constant competitive algorithm uses dynamic programming and requires exponential time.

Cross References

► B-trees
► Degree-Bounded Trees
► Dynamic Trees

Recommended Reading

Based on Wilber [22]'s lower bound, Tango [6] is the first $O(\log\log n)$-competitive binary search tree. Using many of the ideas in Tango and Link-cut Trees [14,19], Multi-Splay Trees [21] generalize the competitive framework to include insertion and deletion. The recommended readings are *Self-adjusting binary search trees* by Sleator and Tarjan, *Lower bounds for accessing binary search trees with rotations* by Wilber, *Dynamic Optimality - Almost* by Demaine, et al, and $O(\log\log n)$ *dynamic competitive binary search tree by Wang, et al.*

1. Blum, A., Chawla, S., Kalai, A.: Static optimality and dynamic search-optimality in lists and trees. Algorithmica **36**, 249–260 (2003)
2. Cole, R.: On the dynamic finger conjecture for splay trees II: The proof. SIAM J. Comput. **30**(1), 44–85 (2000)
3. Cole, R., Mishra, B., Schmidt, J., Siegel, A.: On the dynamic finger conjecture for splay trees I: Splay sorting log *n*-block sequences. SIAM J. Comput. **30**(1), 1–43 (2000)
4. Crane, C.A.: Linear lists and priority queues as balanced binary trees. Technical Report STAN-CS-72-259, Computer Science Dept., Stanford University (1972)
5. Culik II, K., Wood, D.: A note on some tree similarity measures. Inf. Process. Lett. **15**(1), 39–42 (1982)
6. Demaine, E.D., Harmon, D., Iacono, J., Patrascu, M.: Dynamic optimality —almost. SIAM J. Comput. **37**(1), 240–251 (2007)
7. Derryberry, J., Sleator, D.D., Wang, C.C.: A lower bound framework for binary search trees with rotations. Technical Report CMU-CS-05-187, Carnegie Mellon University (2005)
8. Elmasry, A.: On the sequential access theorem and deque conjecture for splay trees. Theor. Comput. Sci. **314**(3), 459–466 (2004)
9. Georgakopoulos, G.F.: How to splay for log log *n*-competitiveness. In: Proc. 4th Int'l Workshop on Experimental and Efficient Algorithms (WEA), pp. 570–579 (2005)
10. Iacono, J.: Key-independent optimality. Algorithmica **42**(1), 3–10 (2005)
11. Knuth, D.E.: Optimum binary search trees. Acta Informatica **1**, 14–25 (1971)
12. Luccio, F., Pagli, L.: On the upper bound on the rotation distance of binary trees. Inf. Process. Lett. **31**(2), 57–60 (1989)
13. Mäkinen, E.: On the rotation distance of binary trees. Inf. Process. Lett. **26**(5), 271–272 (1988)
14. Sleator, D.D., Tarjan, R.E.: Self-adjusting binary search trees. J. ACM **32**(3), 652–686 (1985)
15. Sleator, D.D., Tarjan, R.E., Thurston, W.P.: Rotation distance, triangulations, and hyperbolic geometry. In: Proceedings 18th ACM Symposium on Theory of Computing (STOC), Berkeley, 1986, pp. 122–135
16. Sundar, R.: Twists, turns, cascades, deque conjecture, and scanning theorem. In: Proceedings 30th IEEE Symposium on Foundations of Computer Science (FOCS), pp. 555–559 (1989)
17. Sundar, R.: On the deque conjecture for the splay algorithm. Combinatorica **12**(1), 95–124 (1992)
18. Tarjan, R.: Sequential access in play trees takes linear time. Combinatorica **5**(4), 367–378 (1985)
19. Tarjan, R.E.: Data structures and network algorithms, *CBMS-NSF Reg. Conf. Ser. Appl. Math.*, vol. 44. SIAM, Philadelphia, PA (1983)
20. Wang, C.C.: Multi-splay trees. Ph.D. Thesis, Carnegie Mellon University (2006)
21. Wang, C.C., Derryberry, J., Sleator, D.D.: *O*(log log *n*)-competitive dynamic binary search trees. In: Proc. 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), Miami, 2006, pp. 374–383
22. Wilber, R.: Lower bounds for accessing binary search trees with rotations. SIAM J. Comput. **18**(1), 56–67 (1989)

Online Interval Coloring

1981; Kierstead, Trotter

LEAH EPSTEIN
Department of Math, University of Haifa, Haifa, Israel

Keywords and Synonyms

An extremal problem in recursive combinatorics

Problem Definition

Online interval coloring is a graph coloring problem. In such problems the vertices of a graph are presented one by one. Each vertex is presented in turn, along with a list of its edges in the graph, which are incident to previously presented vertices. The goal is to assign colors (which without loss of generality are assumed to be non-negative integers) to the vertices, so that two vertices which share an edge receive different colors, and the total number of colors used (or alternatively, the largest index of any color that is used) is minimized. The smallest number of colors, for which the graph still admits a valid coloring, is called the chromatic number of the graph.

The interval coloring problem is defined as follows. Intervals on the real line are presented one by one, and the online algorithm must assign each interval a color before the next interval arrives, so that no two intersecting intervals receive the same color. The goal is again to minimize the number of colored used to color any interval. The last problem is equivalent to coloring of interval graphs. These are graphs which have a representation (or realization) where each interval represents a vertex, and two vertices share an edge if and only if they intersect. It is assumed that the interval graph arrives online together with its realization.

Given an interval graph, denote the size of the largest cardinality clique (complete subgraph) in it by ω. Interval graphs have the special property that in a realization, the set of vertices in a clique have a common point in which they all intersect.

Before discussing the online problem, some properties of interval graphs need to be stated. There exists a simple offline algorithm which produces an optimal coloring of interval graphs. An algorithm applies First Fit, if each time it needs to assign a color to an interval, it assigns a smallest index color which still produces a valid coloring. The optimal algorithm simply considers intervals sorted from left to right by their left endpoints, and applies First Fit. Note that the resulting coloring never uses more than ω colors. Indeed, interval graphs are perfect[1].

However, once intervals arrive online in an arbitrary order, it is impossible to design an optimal coloring. Consider a simple example where the two intervals [1,3] and [6,8] are introduced. If they are colored using two distinct colors, this is already sub-optimal, since using the same color for both of them is possible. However, if the sequence of intervals is augmented with [2,5] and [4,7], these two new intervals cannot receive the color of the previous intervals, or the same color for both new intervals. Thus three colors are used, even though a valid coloring using two colors can be designed. Note that even if it is known in advance that the input can be colored using exactly two colors, not knowing whether the additional intervals are as defined above, or alternatively, a single interval [2,7] arrives instead, leads to the usage of three colors instead of only two.

Online coloring is typically hard, which already applies to some simple graph classes such as trees. This is due to the lower bound of $\Omega(\log n)$, given by Gyárfás and Lehel [9] on the competitive ratio of online coloring of trees. There are very few classes for which constant bounds are known. One such class is line graphs, for which Bar-Noy, Motwani and Naor [3] showed that First-Fit is 2-competitive (specifically it uses at most $2 \cdot opt - 1$ colors, where OPT is the number of colors in an optimal coloring), and this is best possible. This result was later generalized to $k \cdot opt - k + 1$ for $(k + 1)$-claw free graphs by [6] (note that line graphs are 3-claw free).

[1] A graph G is perfect if any induced subgraph of G, G' (including G), can be colored using $\omega(G')$ colors, where $\omega(G')$ is the size of the largest cardinality clique in G'. (For any graph, ω is a clear lower bound on its chromatic number).

Key Results

The paper of Kierstead and Trotter [11] provides a solution to the online interval coloring problem. They show that the best possible competitive ratio is 3 which is achieved by an algorithm they design. More accurately, the following theorem is proved in the paper.

Theorem 1 *Given an interval graph which is introduced online, and presented via its realization, any online algorithm uses at least $3\omega - 2$ colors to color the graph, and there exists an algorithm which achieves this bound.*

In the sequel the algorithm and the lower bound construction are described. Note that the algorithm does not need to know ω in advance. Moreover, even though the algorithm is deterministic, it was shown in [12] that the lower bound of 3 on the competitive ratio of online algorithms for interval coloring holds for randomized algorithms as well. Thus [11], gives a complete solution for the problem!

The main idea of the algorithm is creation of "levels". At the time of arrival of an interval, it is classified into a level as follows. Denote by A_k the union of sets of intervals which currently belong to all levels $1, \dots, k$. Intervals are classified so that the largest cardinality clique in A_k is of size k. Thus, A_1 is simply a set of non-intersecting intervals. On arrival of an interval, the algorithm finds the smallest k such that the new interval can join level k, without violating the rule above. It can be shown that each level can be colored using two colors by an offline algorithm. Since the algorithm defined here is online, such a coloring cannot be found in general (see example above). However it is shown in [11] that at most three colors are required for each such level, and a coloring using three colors can be found by applying First Fit on each level (with disjoint sets of colors). Moreover, the first level can always colored using a single color, and ω is equal exactly to the number of levels. Thus a total number of colors, which is at most $3\omega - 2$, is used.

Next, the deterministic lower bound is sketched. The idea is to create levels as in the algorithm. The levels are called phases. Each phase increases the largest clique size by 1, until the value ω is reached. Moreover, every phase (except for the first one) increases the number of colors used by the algorithm by 3.

After each phase was created, some parts of the line are shrunk into single points. Given a point p, that is a result of shrinking an interval $[a, b]$. Every interval presented in the past which is contained in $[a, b]$ is also shrunk into p and therefore such a point inherits a list of colors which no interval that contains it can receive. This is done for simplicity of the proof and means that for a given point p that is the result of shrinking, either contains all intervals that were shrunk into this point, or it has no overlap with any of them.

The sequence construction stops once $3\omega - 2$ colors have been used. Therefore it can be assumed that an initial palette of $3\omega - 3$ colors is given, where these colored can all be used by the algorithm. The ith color ever used is called color number i. As soon as color $3\omega - 2$ is used (even if it is before the construction is completed), the construction stops. The constructed input has a maximum clique is of size (at most) ω.

The sequence starts with introducing a large enough number of intervals N, this is phase 0. Since the algorithm is using at most $3\omega - 3$ colors, this means that there exists a set of $N/(3\omega - 3)$ intervals that share the exact same color. All intervals are shrunk into single points. Later phases result in additional points.

Next, phase $i < \omega$ is defined. The phases are constructed in a way that in the beginning of phase i there is a large enough number of points that contain a given set of $3i - 2$ colors (points of interest). Without loss of generality, assume that these are colors $1 \ldots, 3i - 2$ where the size of the largest clique is i. There exist some other points containing other sets of i colors, or sets of at most $i - 1$ colors. All these points are called void points. At this time, the points of interest are partitioned into consecutive sets of four.

Next, some additional intervals are defined, increasing the size of largest clique by exactly one. Given a set of four points a_1, a_2, a_3, a_4, let b be the leftmost void point on the right hand side of a_1, between a_1 and a_2. If no such point exists, then let $b = (a_1 + a_2)/2$. Similarly, let c be the rightmost void point between a_3 and a_4, and if no such point exists then $c = (a_3 + a_4)/2$. Let d be a point between a_2 and a_3 that is not a void point. The intervals $I_1 = [a_1, (a_1+b)/2]$ and $I_2 = [(c + a_4)/2, a_4]$ are introduced. Clearly none of them may receive one of the currently used $3i - 2$ colors. If they both receive the same new color, the intervals $I_3 = [(a_1 + b)/2, d]$ and $I_4 = [d, (c + a_4)/2]$ are introduced. The interval I_3 intersects with a_2, and with I_1. Therefore it receives an additional color. The second interval I_4 intersects I_3, a_3 and I_2. Therefore a third new color is given to it. If I_1, I_2 receive distinct new colors, the interval $I_5 = [(a_1+b)/2, (c+a_4)/2]$ is introduced. Since I_5 intersects with I_1, I_2, a_2, a_3, it must get a third new color. Every such interval $[a_1, a_4]$ is shrunk into a single point containing $3i + 1$ colors. Since there are less than 3ω colors, and each point uses exactly $3i + 1 < 3\omega$ of them, there are less than $(3\omega)!$ such choices, and a large enough number of points, having the same set of colors, can be picked. The points containing this exact set of colors become the points of interest of the next phase, and the others become void points of the next phase. Points that are void points of previous phases and are not contained in shrunk intervals remain void points. The only points where the new intervals intersect are points with no previous intervals, and therefore the clique size increases by 1 exactly.

At this time phase $i + 1$ can be performed. After phase $\omega - 1$, there are at least $3\omega - 2$ colors in use and the claim is proved. Note that prior to that phase, a minimum number of four points of interest is required.

Applications

In this section, both real-world applications of the problem, and applications of the methods of Kierstead and Trotter [11] to related problems, are discussed.

Many applications arise in various communication networks. The need for connectivity all over the world is rapidly increasing. On the other hand, networks are still composed of very expensive parts. Thus application of optimization algorithms is required in order to save costs.

Consider a network with a line topology that consists of links. Each connection request is for a path between two nodes in the network. The set of requests assigned to a channel must consist of disjoint paths. The goal is to minimize the number of channels (colors) used. A connection request from a to b corresponds to an interval $[a, b]$ and the goal is to minimize the number of required channels to serve all requests.

Another network related application is that if the requests have constant duration c, and all requests have to be served as fast as possible. In this case the colors correspond to time slots, and the total number of colors corresponds to the schedule length.

These are just sample applications, the problem can be described as a scheduling problem as well, and it is clearly of theoretical interest being a natural online graph coloring problem.

Two later studies are of possible interest here, both due to their relevance to the original problem and for the usage of related methods.

The applications in networks stated above raise a generalized problem studied in the recent years. In these applications, it is assumed that once a connection request between two points is satisfied, the channel is blocked at least for the duration of this request. An interesting question, that was raised by Adamy and Erlebach [1], is the following. Assume that a request consists not only of a requested interval, but also from a bandwidth requirement. That is, a customer of a communication channel specifies exactly how much of the channel is needed. Thus, in some cases it is possible to have overlapping requests sharing the same channel. It is required that at every point, the sum of all bandwidth requirements of requests sharing a color cannot exceed the value 1, which is the capacity of the channel. This problem is called *online interval coloring with bandwidth*. In the paper [1], a (large) constant competitive algorithm was designed for the problem. The original interval coloring problem is a special case of this problem where all bandwidth requests are 1. Note that this problem is a generalization of *bin packing* as well, since bin packing is the special case of the problem where all requests have a common point. Azar et al. [2] designed an algorithm of competitive ratio of at most 10 for this problem. This was done by partitioning the requests into four classes based on their bandwidth requirements, and coloring each such class separately. The class of requests with bandwidth in $\left(\frac{1}{2}, 1\right]$ was colored using the basic algorithm of [11], since no two such requests colored with one color can overlap. The two other classes, which are $\left(0, \frac{1}{4}\right]$ and $\left(\frac{1}{4}, \frac{1}{2}\right]$ were colored using adaptations of the algorithm of [11]. Epstein and Levy [7,8] designed improved lower bounds on the competitive ratio, showing that online interval coloring with bandwidth is harder than online interval coloring.

Another problem related to coloring is the *max coloring* problem. In this problem each interval is given a nonnegative weight. Given a coloring, the weight of a color is the maximum weight of any vertex of this color. The goal is to minimize the sum of weights of the used colors. Note that if all weights are 1, max coloring reduces to the graph coloring problem. Pemmaraju, Raman and Varadarajan [13] studied the *offline* max coloring problem for interval graphs. They apply an algorithm which is based on the algorithm of [11]. Thus, they sort the intervals according to their weights, in a monotone nonincreasing order. However, since their algorithm is not online, they exploit the property stated above, that every level is actually 2-colorable, and thus this results in an offline 2-approximation to *max coloring* on interval graphs.

Epstein and Levin [5] studied online max coloring on interval graphs. They design a general reduction which allows to convert algorithms for graph coloring into algorithms for max coloring. The loss in the competitive ratio is a multiplicative factor of 4 for deterministic algorithms, and a factor of $e \approx 2.718$ for randomized algorithms. Thus, using the algorithm of [11] as a black box, they obtained a 12-competitive deterministic algorithm and a $3 \cdot e \approx 8.155$-competitive randomized algorithm. Another result of [5] is lower bound of 4 on the competitive ratio of any randomized algorithm.

Open Problems

Since the paper [11] provided a nice and clean solution to the online interval coloring problem, it does not directly raise open problems. Yet, one related problem is of interest to researchers over the last thirty years, which is the performance of First Fit on this problem. It was shown by Kierstead [10] that First Fit uses at most 40ω colors, thus implying that First Fit has a constant competitive ratio. The quest after the exact competitive ratio was never completed. The best current published results are an upper bound of $10k$ by [13] and a lower bound of $4.4k$ by Chrobak and Slusarek [4]. See [14] for recent developments. It is interesting to note that for online interval coloring with bandwidth, First Fit has an unbounded competitive ratio [1].

Another open problem is to find the best possible competitive ratios for online interval coloring with bandwidth and for max coloring of interval graphs. As stated above, the gap for coloring with bandwidth is currently between $24/7 \approx 3.4286$ by [7,8] and 10 [2], and the gap for max coloring is between 4 and 12 [5].

Recommended Reading

1. Adamy, U., Erlebach, T.: Online coloring of intervals with bandwidth. In: Proc. of the First International Workshop on Approximation and Online Algorithms (WAOA2003), pp. 1–12 (2003)
2. Azar, Y., Fiat, A., Levy, M., Narayanaswamy, N.S.: An improved algorithm for online coloring of intervals with bandwidth. Theor. Comput. Sci. **363**(1), 18–27 (2006)
3. Bar-Noy, A., Motwani, R., Naor, J.: The greedy algorithm is optimal for on-line edge coloring. Inf. Proc. Lett. **44**(5), 251–253 (1992)
4. Chrobak, M., Ślusarek, M.: On some packing problems relating to dynamical storage allocation. RAIRO J. Inf. Theor. Appl. **22**, 487–499 (1988)
5. Epstein, L., Levin, A.: On the max coloring problem. In: Proc. of the Fifth International Workshop on Approximation and Online Algorithms (WAOA2007) (2007), pp. 142–155
6. Epstein, L., Levin, A., Woeginger, G.J.: Graph coloring with rejection. In: Proc. of 14th European Symposium on Algorithms (ESA2006), pp. 364–375. (2006)
7. Epstein, L., Levy, M.: Online interval coloring and variants. In: Proc. of The 32nd International Colloquium on Automata, Languages and Programming (ICALP2005), pp. 602–613. (2005)
8. Epstein, L., Levy, M.: Online interval coloring with packing constraints. In: Proc. of the 30th International Symposium on Mathematical Foundations of Computer Science (MFCS2005), pp. 295–307. (2005)

9. Gyárfás, A., Lehel, J.: Effective on-line coloring of P_5-free graphs. Combinatorica **11**(2), 181–184 (1991)
10. Kierstead, H.A.: The linearity of first-fit coloring of interval graphs. SIAM J. Discret. Math. **1**(4), 526–530 (1988)
11. Kierstead, H.A., Trotter, W.T.: An extremal problem in recursive combinatorics. Congr. Numerantium **33**, 143–153 (1981)
12. Leonardi, S., Vitaletti, A.: Randomized lower bounds for online path coloring. In: Proc. of the second International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM'98), pp. 232–247. (1998)
13. Pemmaraju, S., Raman, R., Varadarajan, K.: Buffer minimization using max-coloring. In: Proc. of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2004), pp. 562–571. (2004)
14. Trotter, W.T.: Current research problems: First Fit colorings of interval graphs. http://www.math.gatech.edu/~trotter/rprob.htm Access date: December 24, 2007.

Online Learning

► Perceptron Algorithm

Online List Update

1985; Sleator, Tarjan

Susanne Albers
Institute for Computer Science, University of Freiburg, Freiburg, Germany

Keywords and Synonyms

Self organizing lists

Problem Definition

The *list update problem* represents a classical online problem and, beside paging, is the first problem that was studied with respect to competitiveness. The list update problem is to maintain a dictionary as an unsorted linear list. Consider a set of items that is represented as a linear linked list. The system is presented with a request sequence σ, where each request is one of the following operations. (1) It can be an *access* to an item in the list, (2) it can be an *insertion* of a new item into the list, or (3) it can be a *deletion* of an item. To access an item, a list update algorithm starts at the front of the list and searches linearly through the items until the desired item is found. To insert a new item, the algorithm first scans the entire list to verify that the item is not already present and then inserts the item at the end of the list. To delete an item, the algorithm scans the list to search for the item and then deletes it.

In serving requests a list update algorithm incurs cost. If a request is an access or a delete operation, then the incurred cost is i, where i is the position of the requested item in the list. If the request is an insertion, then the cost is $n + 1$, where n is the number of items in the list before the insertion. While processing a request sequence, a list update algorithm may rearrange the list. Immediately after an access or insertion, the requested item may be moved at no extra cost to any position closer to the front of the list. These exchanges are called *free exchanges*. Using free exchanges, the algorithm can lower the cost on subsequent requests. At any time two adjacent items in the list may be exchanged at a cost of 1. These exchanges are called *paid exchanges*. The goal is to serve the request sequence so that the total cost is as small as possible.

Of particular interest are list update algorithms that work *online*, i. e. each request is served without knowledge of any future requests. The performance of online algorithms is usually evaluated using *competitive analysis*. Here an online strategy is compared to an optimal offline algorithm that knows the entire request sequence in advance and can serve it with minimum cost. Given a request sequence σ, let $A(\sigma)$ denote the cost incurred by an online algorithm A in serving σ, and let $OPT(\sigma)$ denote the cost incurred by an optimal offline algorithm OPT. Online algorithm A is called c-competitive if there is a constant α such that for all size lists and all request sequences σ, $A(\sigma) \leq c \cdot OPT(\sigma) + \alpha$. The factor c is also called the *competitive ratio*. The competitiveness must hold *for all size lists*.

Key Results

There are three well-known deterministic online algorithms for the list update problem.

Algorithm Move-To-Front: Move the requested item to the front of the list.

Algorithm Transpose: Exchange the requested item with the immediately preceding item in the list.

Algorithm Frequency-Count: Maintain a frequency count for each item in the list. Whenever an item is requested, increase its count by 1. Maintain the list so that the items always occur in nonincreasing order of frequency count.

The formulations of list update algorithms generally assume that a request sequence consists of accesses only. It is obvious how to extend the algorithms so that they can also handle insertions and deletions. On an insertion, the algorithm first appends the new item at the end of the list and then executes the same steps as if the item was requested for the first time. On a deletion, the algorithm first searches for the item and then just removes it.

First consider the algorithms Move-To-Front, Transpose and Frequency-Count. Note that Move-To-Front and Transpose are *memoryless* strategies, i. e. they do not need any extra memory to decide where a requested item should be moved. Thus, from a practical point of view, they are more attractive than Frequency-Count. Sleator and Tarjan [16] analyzed the competitive ratios of the three algorithms.

Theorem 1 ([16]) *The Move-To-Front algorithm is 2-competitive.*

The algorithms Transpose and Frequency-Count are not c-competitive, for any constant c.

Karp and Raghavan [13] developed a lower bound on the competitiveness that can be achieved by deterministic online algorithms. This lower bound implies that Move-To-Front has an optimal competitive ratio.

Theorem 2 ([13]) *Let A be a deterministic online algorithm for the list update problem. If A is c-competitive, then* $c \geq 2$.

An interesting issue is randomization in the list update problem. Against adaptive adversaries, no randomized online algorithm for list update can be better than 2-competitive, see [6,14]. Thus one concentrates on algorithms against oblivious adversaries. Many randomized algorithms for list update have been proposed [1,2,12,14]. The following paragraphs describe the two most important algorithms. Reingold et al. [14] gave a very simple algorithm, called Bit.

Algorithm Bit: Each item in the list maintains a bit that is complemented whenever the item is accessed. If an access causes a bit to change to 1, then the requested item is moved to the front of the list. Otherwise the list remains unchanged. The bits of the items are initialized independently and uniformly at random.

Theorem 3 ([14]) *The Bit algorithm is 1.75-competitive against any oblivious adversary.*

Reingold et al. [14] generalized the Bit algorithm and proved an upper bound of $\sqrt{3} \approx 1.73$ against oblivious adversaries. The best randomized algorithm currently known is a combination of the Bit algorithm and a deterministic 2-competitive online algorithm called Timestamp proposed in [1].

Algorithm Timestamp (TS): Insert the requested item, say x, in front of the first item in the list that precedes x and that has been requested at most once since the last request to x. If there is no such item or if x has not been requested so far, then leave the position of x unchanged.

As an example, consider a list of six items being in the order $L\colon x_1 \rightarrow x_2 \rightarrow x_3 \rightarrow x_4 \rightarrow x_5 \rightarrow x_6$. Suppose that algorithm TS has to serve the second request to x_5 in the request sequence $\sigma = \ldots x_5, x_2, x_2, x_3, x_1, x_1, x_5$. Items x_3 and x_4 were requested at most once since the last request to x_5, whereas x_1 and x_2 were both requested twice. Thus, TS will insert x_5 immediately in front of x_3 in the list. A combination of Bit and TS was proposed by [3].

Algorithm Combination: With probability $\frac{4}{5}$ the algorithm serves a request sequence using Bit, and with probability $\frac{1}{5}$ it serves a request sequence using TS.

Combination achieves the best competitive ratio currently known.

Theorem 4 ([2]) *The algorithm Combination is 1.6-competitive against any oblivious adversary.*

Ambühl et al. [4] presented a lower bound for randomized list update algorithms.

Theorem 5 ([4]) *Let A be a randomized online algorithm for the list update problem. If A is c-competitive against any oblivious adversary, then* $c \geq 1.50084$.

Using techniques from learning theory, Blum et al. [9] recently gave a randomized online algorithm that, for any $\epsilon > 0$, is $(1.6+\epsilon)$-competitive and at the same time $(1+\epsilon)$-competitive against an offline algorithm that is restricted to serving a request sequence with a static list.

So far this entry has considered online algorithms. In the offline variant of the list update problem, the entire request sequence σ is known in advance. Ambühl [3] showed that the offline variant is NP-hard.

Reingold et al. [14] studied an extended cost model, called the P^d model, for the list update problem. In the P^d model there are no free exchanges and each paid exchange costs d. Reingold et al. analyzed deterministic and randomized strategies in this model.

Many of the concepts shown for self-organizing linear lists can be extended to binary search trees. The most popular version of self-organizing binary search trees are the *splay trees* presented by Sleator and Tarjan and the reader is refered to [17].

Regarding the history of the list update problem, prior to competitive analysis, list update algorithms were studied in scenarios where request sequences are generated according to probability distributions. The asymptotic expected cost incurred by an online algorithm in serving a single request was compared to corresponding cost incurred by the optimal static ordering. There exists a large body of literature. Chung et al. [11] showed that, for any distribution, the asymptotic service cost of Move-To-Front is at most $\pi/2$ times that of the optimal ordering.

This bound is tight. Rivest [15] identified distributions on which Transpose performs better than Move-To-Front.

Applications

Linear lists are one possibility for representing a set of items. Certainly, there are other data structures such as balanced search trees or hash tables that, depending on the given application, can maintain a set in a more efficient way. In general, linear lists are useful when the set is small and consists of only a few dozen items. The most important application of list update algorithms are locally adaptive data compression schemes. In fact, Burrows and Wheeler [10] developed a data compression scheme using linear lists that achieves a better compression than Ziv-Lempel based algorithms. Before the description of that algorithm, the next paragraph first presents a data compression scheme given by Bentley et al. [8] that is very simple and easy to implement.

In data compression one is given a string S that shall be compressed, i. e. that shall be represented using fewer bits. The string S consists of symbols, where each symbol is an element of the alphabet $\Sigma = \{x_1, \dots, x_n\}$. The idea of data compression schemes using linear lists it to convert the string S of symbols into a string I of integers. An encoder maintains a linear list of symbols contained in Σ and reads the symbols in the string S. Whenever the symbol x_i has to be compressed, the encoder looks up the current position of x_i in the linear list, outputs this position and updates the list using a list update rule. If symbols to be compressed are moved closer to the front of the list, then frequently occurring symbols can be encoded with small integers. A decoder that receives I and has to recover the original string S also maintains a linear list of symbols. For each integer j it reads from I, it looks up the symbol that is currently stored at position j. Then the decoder updates the list using the same list update rule as the encoder. As list update rule one may use any (deterministic) online algorithm. Clearly, when the string I is actually stored or transmitted, each integer in the string should be coded using a variable length prefix code.

Burrows and Wheeler [10] developed a very effective data compression algorithm using self-organizing lists. The algorithm first applies a reversible transformation to the string S. The purpose of this transformation is to group together instances of a symbol x_i occurring in S. The resulting string S' is then encoded using the Move-To-Front algorithm. More precisely, the transformed string S' is computed as follows. Let m be the length of S. The algorithm first computes the m rotations (cyclic shifts) of S and sorts them lexicographically. Then it extracts the last character of these rotations. The kth symbol of S' is the last symbol of the kth sorted rotation. The algorithm also computes the index J of the original string S in the sorted list of rotations. Burrows and Wheeler gave an efficient algorithm to recover the original string S given only S' and J. The corresponding paper [10] gives a very detailed description of the algorithm and reports of experimental results. On the Calgary Compression Corpus [18], the algorithm outperforms the UNIX utilities compress and gzip and the improvement is 13% and 6%, respectively.

Open Problems

The most important open problem is to determine tight upper and lower bounds on the competitive ratio that can be achieved by randomized online list update algorithms against oblivious adversaries. It is not clear what the true competitiveness is. It is conjectured that the bound is below 1.6. However, as implied by Theorem 5 the performance ratio must be above 1.5.

Experimental Results

Early experimental analyses of the algorithms Move-To-Front, Transpose and Frequency Count were performed by Rivest [15] as well as Bentley and McGeoach [7]. A more recent and extensive experimental study was presented by Bachrach et al. [5]. They implemented and tested a large number of online list update algorithms on request sequences generated by probability distributions and Markov chains as well as on sequences extracted from the Calgary Corpus. It shows that the locality of reference considerably influences the absolute and relative performance of the algorithms. Bachrach et al. also analyzed the various algorithms as data compression strategies.

Recommended Reading

1. Albers, S.: Improved randomized on-line algorithms for the list update problem. SIAM J. Comput. **27**, 670–681 (1998)
2. Albers, S., von Stengel, B., Werchner, R.: A combined BIT and TIMESTAMP algorithm for the list update problem. Inf. Proc. Lett. **56**, 135–139 (1995)
3. Ambühl, C.: Offline list update is NP-hard. In: Proc. 8th Annual European Symposium on Algorithms, pp. 42–51. LNCS, vol. 1879. Springer (2001)
4. Ambühl, C., Gärtner, B., von Stengel, B.: Towards new lower bounds for the list update problem. Theor. Comput. Sci **68**, 3–16 (2001)
5. Bachrach, B., El-Yaniv, R., Reinstädtler, M.: On the competitive theory and practice of online list accessing algorithms. Algorithmica **32**, 201–245 (2002)
6. Ben-David, S., Borodin, A., Karp, R.M., Tardos, G., Wigderson, A.: On the power of randomization in on-line algorithms. Algorithmica **11**, 2–14 (1994)

7. Benteley, J.L., McGeoch, C.C.: Amortized analyses of self-organizing sequential search heuristics. Commun. ACM **28**, 404–411 (1985)
8. Bentley, J.L., Sleator, D.S., Tarjan, R.E., Wei, V.K.: A locally adaptive data compression scheme. Commun. ACM **29**, 320–330 (1986)
9. Blum, A., Chawla, S., Kalai, A.: Static optimality and dynamic search-optimality in lists and trees. In: Proc. 13th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 1–8 (2002)
10. Burrows, M., Wheeler, D.J.: A block-sorting lossless data compression algorithm. DEC SRC Research Report 124, (1994)
11. Chung, F.R.K., Hajela, D.J., Seymour, P.D.: Self-organizing sequential search and Hilbert's inequality. In: Proc. 17th Annual Symposium on the Theory of Computing pp 217–223 (1985)
12. Irani, S.: Two results on the list update problem. Inf. Proc. Lett. **38**, 301–306 (1991)
13. Karp, R. Raghavan, P.: From a personal communication cited in [14]
14. Reingold, N., Westbrook, J., Sleator, D.D.: Randomized competitive algorithms for the list update problem. Algorithmica **11**, 15–32 (1994)
15. Rivest, R.: On self-organizing sequential search heuristics. Commun. ACM **19**, 63–67 (1976)
16. Sleator, D.D., Tarjan, R.E.: Amortized efficiency of list update and paging rules. Commun. ACM **28**, 202–208 (1985)
17. Sleator, D.D., Tarjan, R.E.: Self-adjusting binary search trees. J. ACM **32**, 652–686 (1985)
18. Witten, I.H., Bell, T.: The Calgary/Canterbury text compression corpus. Anonymous ftp from ftp://ftp.cpsc.ucalgary.ca:/pub/text.compression/corpus/text.compression.corpus.tar.Z

Online Paging and Caching

1985–2002; multiple authors

NEAL E. YOUNG
Department of Computer Science,
University of California at Riverside, Riverside, CA, USA

Keywords and Synonyms

Paging; Caching; Weighted caching; Weighted paging; File caching

Problem Definition

A *file-caching* problem instance specifies a cache size k (a positive integer) and a sequence of requests to files, each with a *size* (a positive integer) and a *retrieval cost* (a non-negative number). The goal is to maintain the cache to satisfy the requests while minimizing the retrieval cost. Specifically, for each request, if the file is not in the cache, one must retrieve it into the cache (paying the retrieval cost) and remove other files to bring the total size of files in the cache to k or less. *Weighted caching*, or *weighted paging* is the special case when each file size is 1. *Paging* is the special case when each file size and each retrieval cost is 1. Then the goal is to minimize *cache misses*, or equivalently the *fault rate*.

An algorithm is *online* if its response to each request is independent of later requests. In practice this is generally necessary. Standard worst-case analysis is not meaningful for online algorithms – any algorithm will have some input sequence that forces a retrieval for every request. Yet worst-case analysis can be done meaningfully as follows. An algorithm is $c(h,k)$-*competitive* if on *any* sequence σ the total (expected) retrieval cost incurred by the algorithm using a cache of size k is at most $c(h,k)$ times the *minimum* cost to handle σ with a cache of size h (plus a constant independent of σ). Then the algorithm has *competitive ratio* $c(h,k)$. The study of competitive ratios is called *competitive analysis*. (In the larger context of approximation algorithms for combinatorial optimization, this ratio is commonly called the *approximation ratio*.)

Algorithms

Here are definitions of a number of caching algorithms; first is LANDLORD. LANDLORD gives each file "credit" (equal to its cost) when the file is requested and not in cache. When necessary, LANDLORD reduces all cached file's credits proportionally to file size, then evicts files as they run out of credit.

File-caching algorithm LANDLORD Maintain real value credit[f] with each file f (credit[f] = 0 if f is not in the cache).

When a file g is requested:

```
1. if $g$ is not in the cache:
2.     until the cache has room for $g$:
3.         for each cached file $f$: decrease credit[$f$] by
           $\Delta \cdot$ size[$f$],
4.             where $\Delta = \min_{f \in \text{cache}}$ credit[$f$]/size[$f$].
5.     Evict from the cache any subset of the zero-credit
           files $f$.
6.     Retrieve $g$ into the cache; set credit[$g$] $\leftarrow$ cost($g$).
7. else Reset credit[$g$] anywhere between its current value
   and cost($g$).
```

For weighted caching, file sizes equal 1. GREEDY DUAL is LANDLORD for this special case. BALANCE is the further special case obtained by leaving credit unchanged in line 7.

For paging, files sizes and costs equal 1. FLUSH-WHEN-FULL is obtained by evicting *all* zero-credit files in line 5; FIRST-IN-FIRST-OUT is obtained by leaving credits unchanged in line 7 and evicting the file that entered the cache earliest in line 5; LEAST-RECENTLY-USED is obtained by raising credits to 1 in line 7 and evicting the least-recently requested file in line 5. The MARKING algorithm is obtained by raising credits to 1 in line 7 and evicting a *random* zero-credit file in line 5.

Key Results

This entry focuses on competitive analysis of paging and caching strategies as defined above. Competitive analysis has been applied to many problems other than paging and caching, and much is known about other methods of analysis (mainly empirical or average-case) of paging and caching strategies, but these are outside scope of this entry.

Paging

In a seminal paper, Sleator and Tarjan showed that LEAST-RECENTLY-USED, FIRST-IN-FIRST-OUT, and FLUSH-WHEN-FULL are $k/(k-h+1)$-competitive [13]. Sleator and Tarjan also showed that this competitive ratio is the best possible for any deterministic online algorithm.

Fiat et al. showed that the *Marking* algorithm is $2H_k$-competitive and that no randomized online algorithm is better than H_k-competitive [7]. Here $H_k = 1 + 1/2 + \cdots + 1/k \approx .58 + \ln k$. McGeoch and Sleator gave an optimal H_k-competitive randomized online paging algorithm [12].

Weighted caching

For weighted caching, Chrobak et al. showed that the deterministic online BALANCE algorithm is k-competitive [5]. Young showed that GREEDY DUAL is $k/(k-h+1)$-competitive, and that GREEDY DUAL is a primal-dual algorithm – it generates a solution to the linear-programming dual which proves the near-optimality of the primal solution [15]. Recently Bansal et al. used the primal-dual framework to give an $O(\log k)$-competitive randomized algorithm for weighted caching [1].

File caching

When each cost equals 1 (the goal is to minimize the *number* of retrievals), or when each file's cost equals the file's size (the goal is to minimize the total number of *bytes* retrieved), Irani gave $O(\log^2 k)$-competitive randomized online algorithms [8].

For general file caching, Irani and Cao showed that a restriction of LANDLORD is k-competitive [4]. Independently, Young showed that LANDLORD is $k/(k-h+1)$-competitive [15].

Other theoretical models

Practical performance can be better than the worst case studied in competitive analysis. Refinements of the model have been proposed to increase realism. Borodin et al. [3], to model locality of reference, proposed the *access-graph* model (see also [9,10]). Koutsoupias and Papadimitriou proposed the *comparative ratio* (for comparing classes of online algorithms directly) and the *diffuse-adversary model* (where the adversary chooses requests probabilistically subject to restrictions) [11]. Young showed that any $k/(k-h+1)$-competitive algorithm is also *loosely* $O(1)$-competitive: for any fixed $\varepsilon, \delta > 0$, on any sequence, for all but a δ-fraction of cache sizes k, the algorithm either is $O(1)$-competitive or pays at most ε times the sum of the retrieval costs [15].

Analyses of deterministic algorithms

Here is a competitive analysis of GREEDY DUAL for weighted caching.

Theorem 1 GREEDY DUAL *is $k/(k-h+1)$-competitive for weighted caching.*

Proof Here is an amortized analysis (in the spirit of Sleator and Tarjan, Chrobak et al., and Young; see [14] for a different primal-dual analysis). Define potential

$$\Phi = (h-1)\cdot \sum_{f\in \mathrm{GD}} \mathrm{credit}[f] + k\cdot \sum_{f\in \mathrm{OPT}} \Big(\mathrm{cost}(f) - \mathrm{credit}[f]\Big),$$

where GD and OPT denote the current caches of GREEDY DUAL and OPT (the optimal off-line algorithm that manages the cache to minimize the total retrieval cost), respectively. After each request, GREEDY DUAL and OPT take (some subset of) the following steps in order.

OPT **evicts a file** f: Since $\mathrm{credit}[f] \le \mathrm{cost}(f)$, Φ cannot increase.

OPT **retrieves requested file** g: OPT pays $\mathrm{cost}(g)$; Φ increases by at most $k\,\mathrm{cost}(g)$.

GREEDY DUAL **decreases** $\mathrm{credit}[f]$ **for all** $f \in \mathrm{GD}$: The cache is full and the requested file is in OPT but not yet in GD. So $|\mathrm{GD}| = k$ and $|\mathrm{OPT} \cap \mathrm{GD}| \le h-1$. Thus, the total decrease in Φ is $\Delta[(h-1)|\mathrm{GD}| - k\,|\mathrm{OPT}\cap\mathrm{GD}|] \ge \Delta[(h-1)k - k(h-1)] = 0$.

GREEDY DUAL **evicts a file** f: Since $\mathrm{credit}[f] = 0$, Φ is unchanged.

GREEDY DUAL **retrieves requested file** g **and sets** $\mathrm{credit}[g]$ **to** $\mathrm{cost}(g)$: GREEDY DUAL pays $c = \mathrm{cost}(g)$. Since g was not in GD but is in OPT, $\mathrm{credit}[g] = 0$ and Φ decreases by $-(h-1)c + k\,c = (k-h+1)c$.

GREEDY DUAL **resets** $\mathrm{credit}[g]$ **between its current value and** $\mathrm{cost}(g)$: Since $g \in \mathrm{OPT}$ and $\mathrm{credit}[g]$ only increases, Φ decreases.

So, with each request: (1) when OPT retrieves a file of cost c, Φ increases by at most kc; (2) at no other time does Φ increase; and (3) when GREEDY DUAL retrieves a file of cost c, Φ decreases by at least $(k - h + 1)c$. Since initially $\Phi = 0$ and finally $\Phi \geq 0$, it follows that GREEDY DUAL's total cost times $k - h + 1$ is at most OPT's cost times k.

Extension to file caching

Although the proof above easily extends to LANDLORD, it is more informative to analyze LANDLORD via a *general reduction* from file caching to weighted caching:

Corollary 1 LANDLORD *is* $k/(k - h + 1)$*-competitive for file caching.*

Proof Let W be any deterministic c-competitive weighted-caching algorithm. Define file-caching algorithm F_W as follows. Given request sequence σ, F_W simulates W on weighted-caching sequence σ' as follows. For each file f, break f into size(f) "pieces" $\{f_i\}$ each of size 1 and cost cost(f)/size(f). When f is requested, give a batch $(f_1, f_2, \dots, f_s)^{N+1}$ of requests for pieces to W. Take N large enough so W has all pieces $\{f_i\}$ cached after the first sN requests of the batch.

Assume that W *respects equivalence*: after each batch, for every file f, all or none of f's pieces are in W's cache. After each batch, make F_W update its cache correspondingly to $\{f\colon f_i \in \text{cache}(W)\}$. F_W's retrieval cost for σ is at most W's retrieval cost for σ', which is at most $c\,\text{OPT}(\sigma')$, which is at most $c\,\text{OPT}(\sigma')$. Thus, F_W is c-competitive for file caching.

Now, observe that GREEDY DUAL can be made to respect equivalence. When GREEDY DUAL processes a batch of requests $(f_1, f_2, \dots, f_s)^{N+1}$ resulting in retrievals, for the last s requests, make GREEDY DUAL set credit$[f_i]$ = cost(f_i) = cost$(f)/s$ in line 7. In general, restrict GREEDY DUAL to raise credits of equivalent pieces f_i equally in line 7. After each batch the credits on equivalent pieces f_i will be the same. When GREEDY DUAL evicts a piece f_i, make GREEDY DUAL evict all other equivalent pieces f_j (all will have zero credit).

With these restrictions, GREEDY DUAL respects equivalence. Finally, taking W to be GREEDY DUAL above, F_W is LANDLORD.

Analysis of the randomized MARKING algorithm

Here is a competitive analysis of the MARKING algorithm.

Theorem 2 *The* MARKING *algorithm is* $2H_k$*-competitive for paging.*

Proof Given a paging request sequence σ, partition σ into contiguous *phases* as follows. Each phase starts with the request after the end of the previous phase and continues as long as possible subject to the constraint that it should contain requests to at most k distinct pages. (Each phase starts when the algorithm runs out of zero-credit files and reduces all credits to zero.)

Say a request in the phase is *new* if the item requested was not requested in the previous phase. Let m_i denote the number of new requests in the ith phase. During phases $i-1$ and i, $k + m_i$ distinct files are requested. OPT has at most k of these in cache at the start of the $i-1$st phase, so it will retrieve at least m_i of them before the end of the ith phase. So OPT's total cost is at least $\max\left\{\sum_i m_{2i}, \sum_i m_{2i+1}\right\} \geq \sum_i m_i/2$.

Say a non-new request is *redundant* if it is to a file with credit 1 and non-redundant otherwise. Each new request costs the MARKING algorithm 1. The jth non-redundant request costs the MARKING algorithm at most $m_i/(k - j + 1)$ in expectation because, of the $k - j + 1$ files that if requested would be non-redundant, at most m_i are not in the cache (and each is equally likely to be in the cache). Thus, in expectation MARKING pays at most $m_i + \sum_{j=1}^{k-m_i} m_i/(k - j + 1) \leq m_i H_k$ for the phase, and at most $H_k \sum_i m_i$ total.

Applications

Variants of GREEDY DUAL and LANDLORD have been incorporated into file-caching software such as Squid [6].

Experimental Results

For a study of competitive ratios on practical inputs, see for example [4,6,14].

Cross References

▶ Algorithm DC-Tree for k Servers on Trees
▶ Alternative Performance Measures in Online Algorithms
▶ Online List Update
▶ Price of Anarchy
▶ Work-Function Algorithm for k Servers

Recommended Reading

1. Bansal, N., Buchbinder, N., Naor, J.: A primal-dual randomized algorithm for weighted paging. Proceedings of 48th Annual IEEE Symposium on Foundations of Computer Science, pp. 507–517 (2007)
2. Borodin, A., El-Yaniv, R.: Online computation and competitive analysis. Cambridge University Press, New York (1998)

3. Borodin, A., Irani, S., Raghavan, P., Schieber B.: Competitive paging with locality of reference. J. Comput. Syst. Sci. **50**(2), 244–258 (1995)
4. Cao, P., Irani, S.: Cost-aware WWW proxy caching algorithms. In: USENIX Symposium on Internet Technologies and Systems, Monterey, December 1997
5. Chrobak, M., Karloff, H., Payne, T.H., Vishwanathan, S.: New results on server problems. SIAM J. Discret. Math. **4**(2), 172–181 (1991)
6. Dilley, J., Arlitt, M., Perret, S.: Enhancement and validation of Squid's cache replacement policy. Hewlett-Packard Laboratories Technical Report HPL-1999–69 (1999)
7. Fiat, A., Karp, R.M., Luby, M., McGeoch, L.A., Sleator, D.D., Young, N.E.: Competitive paging algorithms. J. Algorithms **12**(4), 685–699 (1991)
8. Irani, S.: Page replacement with multi-size pages and applications to Web caching. Algorithmica **33**(3), 384–409 (2002)
9. Irani, S., Karlin, A.R., Phillips, S.: Strongly competitive algorithms for paging with locality of reference. SIAM J. Comput. **25**(3), 477–497 (1996)
10. Karlin, A.R., Phillips, S.J., Raghavan, P.: Markov paging. SIAM J. Comput. **30**(3), 906–922 (2000)
11. Koutsoupias, E., Papadimitriou, C.H.: Beyond competitive analysis. SIAM J. Comput. **30**(1), 300–317 (2000)
12. McGeoch, L.A., Sleator, D.D.: A strongly competitive randomized paging algorithm. Algorithmica **6**, 816–825 (1991)
13. Sleator, D.D., Tarjan, R.E.: Amortized efficiency of list update and paging rules. Commun. ACM **28**(2), 202–208 (1985)
14. Young, N.E.: The *k*-server dual and loose competitiveness for paging. Algorithmica **11**(6), 525–541 (1994)
15. Young, N.E.: On-line file caching. Algorithmica **33**(3), 371–383 (2002)

Online Scheduling

► List Scheduling
► Load Balancing

Optimal Probabilistic Synchronous Byzantine Agreement
1988; Feldman, Micali

Juan Garay
Bell Labs, Murray Hill, NJ, USA

Keywords and Synonyms

Distributed consensus; Byzantine generals problem

Problem Definition

The Byzantine agreement problem (BA) is concerned with multiple processors (parties, "players") all starting with some initial value, agreeing on a common value, despite the possible disruptive or even malicious behavior of some them. BA is a fundamental problem in fault-tolerant distributed computing and secure multi-party computation.

The problem was introduced by Pease, Shostak and Lamport in [18], who showed that the number of faulty processors must be less than a third of the total number of processors for the problem to have a solution. They also presented a protocol matching this bound, which requires a number of communication rounds proportional to the number of faulty processors—exactly $t + 1$, where t is the number of faulty processors. Fischer and Lynch [10] later showed that this number of rounds is necessary in the worst-case run of any deterministic BA protocol. Furthermore, the above assumes that communication takes place in synchronous rounds. Fischer, Lynch and Patterson [11] proved that no completely asynchronous BA protocol can tolerate even a single processor with the simplest form of misbehavior—namely, ceasing to function at an arbitrary point during the execution of the protocol ("crashing").

To circumvent the above-mentioned lower bound on the number of communication rounds and impossibility result, respectively, researchers beginning with Ben-Or [1] and Rabin [19], and followed by many others (e. g., [3,5]) explored the use of randomization. In particular, Rabin showed that linearly resilient BA protocols in expected *constant* rounds were possible, provided that all the parties have access to a "common coin" (i. e., a common source of randomness)—essentially, the value of the coin can be adopted by the non-faulty processors in case disagreement at any given round is detected, a process that is repeated multiple times. This line of research culminated in the unconditional (or information-theoretic) setting with the work of Feldman and Micali [9], who showed an efficient (i. e., polynomial-time) probabilistic BA protocol tolerating the maximal number of faulty processors[1] that runs in expected constant number of rounds. The main achievement of the Feldman–Micali work is to show how to obtain a shared random coin with constant success probability in the presence of the maximum allowed number of misbehaving parties "from scratch".

Randomization has also been applied to BA protocols in the computational (or cryptographic) setting and for weaker failure models. See [6] for an early survey on the subject.

Notations

Consider a set $\mathcal{P} = \{P_1, P_2, \cdots, P_n\}$ of processors (probabilistic polynomial-time Turing machines) out of which t, $t < n$ may not follow the protocol, and even collude and behave in arbitrary ways. These processors are called

[1]Karlin and Yao [14] showed that the maximum number of faulty processors for probabilistic BA is also $t < \frac{n}{3}$, where n is the total number of processors.

faulty; it is useful to model the faulty processors as being coordinated by an adversary, sometimes called a *t-adversary*.

For $1 \le i \le n$, let b_i, $b_i \in \{0, 1\}$ denote party P_i's initial value. The work of Feldman and Micali considers the problem of designing a probabilistic BA protocol in the model defined below.

System Model

The processors are assumed to be connected by point-to-point private channels. Such a network is assumed to be synchronous, i. e., the processors have access to a global clock, and thus the computation of all processors can proceed in a lock-step fashion. It is customary to divide the computation of a synchronous network into *rounds*. In each round, processors send messages, receive messages, and perform some local computation.

The t-adversary is computationally unbounded, *adaptive* (i. e., it chooses which processors to corrupt on the fly), and decides on the messages the faulty processors send in a round depending on the messages sent by the non-faulty processors in all previous rounds, including the current round (this is called a *rushing* adversary).

Given the model above, the goal is to solve the problem stated in the ► Byzantine Agreement; that is, for every set of inputs and any behavior of the faulty processors, to have the non-faulty processors output a common value, subject to the additional condition that if they all start the computation with the same initial value, then that should be the output value. The difference with respect to the other entry is that, thanks to randomization, BA protocols here run in expected constant rounds.

Problem 1 (BA)
Input: *Each processor P_i, $1 \le i \le n$, has bit b_i.*
Output: *Eventually, each processor P_i, $1 \le i \le n$, outputs bit d_i satisfying the following two conditions:*

- Agreement: *For any two non-faulty processors P_i and P_j, $d_i = d_j$.*
- Validity: *If $b_i = b_j = b$ for all non-faulty processors P_i and P_j, then $d_i = b$ for all non-faulty processors P_i.*

In the above definition input and output values are from $\{0, 1\}$. This is without loss of generality, since there is a simple two-round transformation that reduces a multi-valued agreement problem to the binary problem [20].

Key Results

Theorem 1 *Let $t < \frac{n}{3}$. Then there exists a polynomial-time BA protocol running in expected constant number of rounds.*

The number of rounds of the Feldman–Micali BA protocol is expected constant, but there is no bound in the worst case; that is, for every r, the probability that the protocol proceeds for more than r rounds is very small, yet greater than 0—in fact, equal to $2^{-O(r)}$. Further, the non-faulty processors may not terminate in the same round.[2]

The Feldman–Micali BA protocol assumes synchronous rounds. As mentioned above, one of the motivations for the use of randomization was to overcome the impossibility result due to Fischer, Lynch and Paterson [11] of BA in asynchronous networks, where there is no global clock, and the adversary is also allowed to schedule the arrival time of a message sent to a non-faulty processor (of course, faulty processors may not send any message(s)). In [8], Feldman mentions that the Feldman–Micali BA protocol can be modified to work on asynchronous networks, at the expense of tolerating $t < \frac{n}{4}$ faults. In [4], Canetti and Rabin present a probabilistic asynchronous BA protocol for $t < \frac{n}{3}$ that differs from the Feldman–Micali approach in that it is a Las Vegas protocol—i. e., it has non-terminating runs, but when it terminates, it does so in constant expected rounds.

Applications

There exists a one-to-one correspondence, possibility- and impossibility-wise between BA in the unconditional setting as defined above and a formulation of the problem called the "Byzantine generals" by Lamport, Shostak and Pease [16], where there is a distinguished source among the parties sending a value, call it b_s, and the rest of the parties having to agree on it. The Agreement condition remains unchanged; the Validity condition becomes

- Validity: If the source is non-faulty, then $d_i = b_s$ for all non-faulty processors P_i.

A protocol for this version of the problem realizes a functionality called a "broadcast channel" on a network with only point-to-point connectivity. Such a tool is very useful in the context of cryptographic protocols and secure multi-party computation [12]. Probabilistic BA is particularly relevant here, since it provides a constant-round implementation of the functionality. In this respect, without any optimizations, the reported actual number of expected rounds of the Feldman–Micali BA protocol is at most 57.

[2]Indeed, it was shown by Dwork and Moses [7] that at least $t + 1$ rounds are necessary for simultaneous termination. In [13], Goldreich and Petrank combine "the best of both worlds" by showing a BA protocol running in expected constant number of rounds which always terminates within $t + O(\log t)$ rounds.

Recently, Katz and Koo [15] presented a probabilistic BA protocol with an expected number of rounds at most 23.

BA has many other applications. Refer to the ▶ Byzantine Agreement, as well as to, e. g., [17] for further discussion of other application areas.

Cross References

▶ Asynchronous Consensus Impossibility
▶ Atomic Broadcast
▶ Byzantine Agreement
▶ Randomized Energy Balance Algorithms in Sensor Networks

Recommended Reading

1. Ben-Or, M.: Another advantage of free choice: Completely asynchronous agreement protocols. In: Proc. 22nd Annual ACM Symposium on the Principles of Distributed Computing, 1983, pp. 27–30
2. Ben-Or, M., El-Yaniv, R.: Optimally-resilient interactive consistency in constant time. Distrib. Comput. **16**(4), 249–262 (2003)
3. Bracha, G.: An $O(\log n)$ expected rounds randomized Byzantine generals protocol. J. Assoc. Comput. Mach. **34**(4), 910–920 (1987)
4. Canetti, R., Rabin, T.: Fast asynchronous Byzantine agreement with optimal resilience. In: Proceedings of the 25th Annual ACM Symposium on the Theory of Computing, San Diego, California, 16–18 May 1993, pp. 42–51
5. Chor, B., Coan, B.: A simple and efficient randomized Byzantine agreement algorithm. IEEE Trans. Softw. Eng. **SE-11**(6), 531–539 (1985)
6. Chor, B., Dwork, C.: Randomization in Byzantine Agreement. Adv. Comput. Res. **5**, 443–497 (1989)
7. Dwork, C., Moses, Y.: Knowledge and Common Knowledge in a Byzantine Environment: Crash Failures. Inf. Comput. **88**(2), 156–186 (1990). Preliminary version in TARK'86
8. Feldman, P.: Optimal Algorithms for Byzantine Agreement. Ph. D. thesis, MIT (1988)
9. Feldman, P., Micali, S.: An optimal probabilistic protocol for synchronous Byzantine agreement. SIAM J. Comput. **26**(4), 873–933 (1997). Preliminary version in STOC'88
10. Fischer, M.J., Lynch, N.A.: A Lower Bound for the Time to Assure Interactive Consistency. Inf. Process. Lett. **14**(4), 183–186 (1982)
11. Fischer, M.J., Lynch, N.A., Paterson, M.S.: Impossibility of distributed consensus with one faulty processor. J. ACM **32**(2), 374–382 (1985)
12. Goldreich, O.: Foundations of Cryptography, Volumes 1 and 2. Cambridge University Press, Cambridge (2001), (2004)
13. Goldreich, O., Petrank, E.: The Best of Both Worlds: Guaranteeing Termination in Fast Randomized Agreement Protocols. Inf. Process. Lett. **36**(1), 45–49 (1990)
14. Karlin, A., Yao, A.C.: Probabilistic lower bounds for the byzantine generals problem. Unpublished manuscript
15. Katz, J., Koo, C.: On Expected Constant-Round Protocols for Byzantine Agreement. In: Proceedings of Advances in Cryptology–CRYPTO 2006, Santa Barbara, California, August 2006, pp. 445–462. Springer, Berlin Heidelberg New York (2006)
16. Lamport, L., Shostak, R., Pease, M.: The Byzantine generals problem. ACM Trans. Program. Lang. Syst. **4**(3), 382–401 (1982)
17. Lynch, N.: Distributed Algorithms, Morgan Kaufmann, San Francisco (1996)
18. Pease, M., Shostak, R., Lamport, L.: Reaching agreement in the presence of faults. J. ACM **27**(2), 228–234 (1980)
19. Rabin, M.: Randomized Byzantine Generals. In: Proc. 24th Anual IEEE Symposium on Foundations of Computer Science, 1983, pp. 403–409
20. Turpin, R., Coan, B.A.: Extending binary Byzantine Agreement to multivalued Byzantine Agreement. Inf. Process. Lett. **18**(2), 73–76 (1984)

Optimal Radius

▶ Distance-Based Phylogeny Reconstruction (Optimal Radius)

Optimal Stable Marriage

1987; Irving, Leather, Gusfield

ROBERT W. IRVING
Department of Computing Science,
University of Glasgow, Glasgow, UK

Keywords and Synonyms

Optimal stable matching

Problem Definition

The classical *Stable Marriage problem* (SM), first studied by Gale and Shapley [5], is introduced in ▶ Stable Marriage. An instance of SM comprises a set $\mathcal{M} = \{m_1, \dots, m_n\}$ of n men and a set $\mathcal{W} = \{w_1, \dots, w_n\}$ of n women, and for each person a *preference list*, which is a total order over the members of the opposite sex. A man's (respectively woman's) preference list indicates his (respectively her) strict order of preference over the women (respectively men). A matching M is a set of n man–woman pairs in which each person appears exactly once. If the pair (m,w) is in the matching M then m and w are *partners* in M, denoted by $w = M(m)$ and $m = M(w)$. Matching M is stable if there is no man m and woman w such that m prefers w to $M(m)$ and w prefers m to $M(w)$.

The key result established in [5] is that at least one stable matching exists for every instance of SM. In general there may be many stable matchings, so the question arises as to what is an appropriate definition for the 'best' stable matching, and how such a matching may be found.

Gale and Shapley described an algorithm to find a stable matching for a given instance of SM. This algorithm may be applied either from the men's side or from the women's side. In the former case, it finds the so-called *man-optimal* stable matching, in which each man has the best partner, and each woman the worst partner, that is possible in any stable matching. In the latter case, the *woman-optimal* stable matching is found, in which these properties are interchanged. For some instances of SM, the man-optimal and woman-optimal stable matchings coincide, in which case this is the unique stable matching. In general however, there may be many other stable matchings between these two extremes. Knuth [13] was first to show that the number of stable matchings can grow exponentially with n.

Because of the imbalance inherent, in general, in the man-optimal and woman-optimal solutions, several other notions of optimality in SM have been proposed.

A stable matching M is *egalitarian* if the sum

$$\sum_i r(m_i, M(m_i)) + \sum_j r(w_j, M(w_j))$$

is minimized over all stable matchings, where $r(m,w)$ represents the rank, or position, of w in m's preference list, and similarly for $r(w,m)$. An egalitarian stable matching incorporates an optimality criterion that does not overtly favor the members of one sex – though it is easy to construct instances having many stable matchings in which the unique egalitarian stable matching is in fact the man (or woman) optimal.

A stable matching M is *minimum regret* if the value $\max(r(p, M(p))$ is minimized over all stable matchings, where the maximum is taken over all persons p. A minimum regret stable matching involves an optimality criterion based on the least happy member of the society, but again, minimum regret can coincide with man-optimal or woman-optimal in some cases, even when there are many stable matchings.

A stable matching is *rank-maximal* (or *lexicographically maximal*) if, among all stable matchings, the largest number of people have their first choice partner, and subject to that, the largest number have their second choice partner, and so on.

A stable matching M is *sex-equal* if the difference

$$\left|\sum_i r(m_i, M(m_i)) - \sum_j r(w_j, M(w_j))\right|$$

is minimized over all stable matchings. This definition is an explicit attempt to ensure that one sex is treated no more favorably than the other, subject to the overriding criterion of stability.

In the *weighted* stable marriage problem (WSM), each person has, as before, a strictly ordered preference list, but the entries in this list have associated costs or weights – $wt(m,w)$ represents the weight associated with woman w in the preference list of man m, and likewise for $wt(w,m)$. It is assumed that the weights are strictly increasing along each preference list.

A stable matching M in an instance of WSM is *optimal* if

$$\sum_i wt(m_i, M(m_i)) + \sum_j wt(w_j, M(w_j))$$

is minimized over all stable matchings.

A stable matching M in an instance of WSM is *balanced* if

$$\max\left(\sum_i wt(m_i, M(m_i)), \sum_j wt(w_j, M(w_j))\right)$$

is minimized over all stable matchings.

These same forms of optimality may be defined in the more general context of the Stable Marriage problem with Incomplete Preference Lists (SMI) –see ▶ Stable Marriagefor a formal definition of this problem.

Again as described in ▶ Stable Marriage, the *Stable Roommates* problem (SR) is a non-bipartite generalization of SM, also introduced by Gale and Shapley [5]. In contrast to SM, an instance of SR may or may not admit a stable matching. Irving [9] gave the first polynomial-time algorithm to determine whether an SR instance admits a stable matching, and if so to find one such matching.

There is no concept of man or woman optimal in the SR context, and nor is there any analogue of sex-equal or balanced matching. However, the other forms of optimality introduced above can be defined also for instances of SR and WSR (Weighted Stable Roommates).

A comprehensive treatment of many aspects of the Stable Marriage problem, as of 1989, appears in the monograph of Gusfield and Irving [6].

Key Results

The key to providing efficient algorithms for the various kinds of optimal stable matching is an understanding of the algebraic structure underlying an SM instance, and the discovery of methods to exploit this structure. Knuth [13] attributes to Conway the observation that the set of stable matchings for an SM instance forms a distributive lattice under a natural dominance relation. Irving and Leather [11] characterized this lattice in terms of so-called *rotations* – essentially minimal differences between lattice elements – which can be efficiently computed directly from the preference lists. The rotations form a natural partial order, the *rotation poset*, and there is a one-to-one correspondence between the stable matchings and the closed subsets of the rotation poset.

Building on these structural results, Gusfield [8] gave a $O(n^2)$ algorithm to find a minimum-regret stable matching, improving an earlier $O(n^4)$ algorithm described by Knuth [13] and attributed to Selkow. Irving et al. [10] showed how application of network flow methods to the rotation poset yield efficient algorithms for egalitarian and rank-maximal stable matchings, as well as for an optimal stable matching in WSM. These algorithms have complexities $O(n^4)$, $O(n^5 \log n \log n)$ and $O(n^4 \log n)$ respectively. Subsequently, by using an interpretation of a stable marriage instance as an instance of 2-SAT, and with the aid of a faster network flow algorithm exploiting the special structure of networks representing SM instances, Feder [3,4] reduced these complexities to $O(n^3)$, $O(n^{3.5})$ and $O(\min(n, \sqrt{K})n^2 \log(K/n^n + 2))$ respectively, where K is the weight of an optimal solution.

By way of contrast, and perhaps surprisingly, the problems of finding a sex-equal stable matching for an instance of SM and of finding a balanced stable matching for an instance of WSM have been shown to be NP-hard [2,12].

The following theorem summarizes the current state of knowledge regarding the various flavors of optimal stable matching in SM and WSM.

Theorem 1 *For an instance of SM:*

(i) A minimum regret stable matching can be found in $O(n^2)$ time.

(ii) An egalitarian stable matching can be found in $O(n^3)$ time.

(iii) A rank-maximal stable matching can be found in $O(n^{3.5})$ time.

(iv) The problem of finding a sex-equal stable matching is NP-hard.

For an instance of WSM:

(v) An optimal stable matching can be found in $O(\min(n, \sqrt{K})n^2 \log(K/n^2 + 2))$ time, where K is the weight of an optimal solution.

(vi) The problem of finding a balanced stable matching is NP-hard, but can be approximated within a factor of 2 in $O(n^2)$ time.

Among related problems that can also be solved efficiently by exploitation of the rotation structure of an instance of SM are the following [8]:

- all stable pairs, i. e., pairs that belong to at least one stable matching, can be found in $O(n^2)$ time;
- all stable matchings can be enumerated in $O(n^2 + kn)$ time, where k is the number of such matchings.

Results analogous to those of Theorem 1 are known for the more general SMI problem. In the case of the Stable Roommates problem (SR), some of these problems appear to be harder, as summarized in the following theorem.

Theorem 2 *For an instance of SR:*

(i) A minimum regret stable matching can be found in $O(n^2)$ time [7].

(ii) The problem of finding an egalitarian stable matching is NP-hard. It can be approximated in polynomial time within a factor of α if and only if minimum vertex cover can be approximated within α [1,2].

For an instance of WSR (weighted stable roommates):

(iii) The problem of finding an optimal stable matching is NP-hard, but can be approximated within a factor of 2 in $O(n^2)$ time [3].

Applications

The best known and most important applications of stable matching algorithms are in centralized matching schemes in the medical and educational domains. ► Hospitals / Residents Problem includes a summary of some of these applications.

Open Problems

There remains the possibility of improving the complexity bounds for some of the optimization problems discussed, and for finding better polynomial-time approximation algorithms for the various NP-hard problems.

Cross References

► Hospitals/Residents Problem
► Ranked Matching
► Stable Marriage and Discrete Convex Analysis
► Stable Marriage with Ties and Incomplete Lists
► Stable Partition Problem

Recommended Reading

1. Feder, T.: A new fixed point approach for stable networks and stable marriages. In: Proceedings of 21st ACM Symposium on Theory of Computing, pp. 513–522, Theory of Computing, Seattle WA, May 1989, pp. 513–522, ACM, New York (1989)
2. Feder, T.: Stable networks and product graphs. Ph. D. thesis, Stanford University (1991)
3. Feder, T.: A new fixed point approach for stable networks and stable marriages. J. Comput. Syst. Sci. **45**, 233–284 (1992)
4. Feder, T.: Network flow and 2-satisfiability. Algorithmica **11**, 291–319 (1994)
5. Gale, D., Shapley, L.S.: College admissions and the stability of marriage. Am. Math. Mon. **69**, 9–15 (1962)

6. Gusfield, D., Irving, R.W.: The Stable Marriage Problem: Structure and Algorithms. MIT Press, Cambrigde MA (1989)
7. Gusfield, D.: The structure of the stable roommate problem: efficient representation and enumeration of all stable assignments. SIAM J. Comput. **17**(4), 742–769 (1988)
8. Gusfield, D.: Three fast algorithms for four problems in stable marriage. SIAM J. Comput. **16**(1), 111–128 (1987)
9. Irving, R.W.: An efficient algorithm for the stable roommates problem. J. Algorithms **6**, 577–595 (1985)
10. Irving, R.W., Leather, P., Gusfield, D.: An efficient algorithm for the "optimal stable" marriage. J. ACM **34**(3), 532–543 (1987)
11. Irving, R.W., Leather, P.: The complexity of counting stable marriages. SIAM J. Comput. **15**(3), 655–667 (1986)
12. Kato, A.: Complexity of the sex-equal stable marriage problem. Jpn. J. Ind. Appl. Math. **10**, 1–19 (1993)
13. Knuth, D.E.: Mariages Stables. Les Presses de L'Université de Montréal (1976)

P

P2P

2001; Stoica, Morris, Karger, Kaashoek, Balakrishnan

Dahlia Malkhi
Microsoft, Silicon Valley Campus,
Mountain View, CA, USA

Keywords and Synonyms

Peer to peer; Overlay; Overlay network; DHT; Distributed hash table; CDN; Content delivery network; File sharing; Resource sharing

Problem Definition

This problem is concerned with efficiently designing a serverless infrastructure for a federation of hosts to store, index and locate information, and for efficient data dissemination among the hosts. The key services of peer-to-peer (P2P) overlay networks are:

1. A keyed lookup protocol locates information at the server(s) that hold it.
2. Data store, update and retrieve operations maintain a distributed persistent data repository.
3. Broadcast and multicast support information dissemination to multiple recipients.

Because of their symmetric, serverless nature, these networks are termed *P2P* networks. Below, we often refer to hosts participating in the network as *peers*.

The most influential mechanism in this area is *consistent hashing*, pioneered in a paper by Karger et al. [21]. The idea is roughly the following. Frequently, a good way of arranging a lookup directory is a hash table, giving a fast $O(1)$-complexity data access. In order to scale and provide highly available lookup services, we partition the hash table and assign different chunks to different servers. So, for example, if the hash table has entries 1 through n, and there are k participating servers, we can have each server select a virtual identifier from 1 to n at random. Server i will then be responsible for key values that are closer to i than to any other server identifier. With a good randomization of the hash keys, we can have a more or less balanced distribution of information between our k servers. In expectation, each server will be responsible for (n/k) keys. Furthermore, the departure/arrival of a server perturbs only one or two other servers with adjacent virtual identifiers.

A network of servers that implement consistent hashing is called a *distributed hash table* (DHT). Many current-generation resource sharing networks, and virtually all academic research projects in the area, are built around a DHT idea.

The challenge in maintaining DHTs is two-fold:

Overlay routing Given a hash key i, and starting from any node r in the network, the problem is to find the server s whose key range contains i. The key name i bears no relation to any real network address, such as the IP address of a node, and therefore we cannot use the underlying IP infrastructure to locate s. An overlay routing network links the nodes, and provides them with a routing protocol, such that r can route toward s using the routing target i.

Dynamic maintenance DHTs must work in a highly dynamic environment in which the size of the network is not known a priori, and where there are no permanent servers for maintaining either the hash function or the overlay network (all servers are assumed to be ephemeral). This is especially acute in P2P settings, where the servers are transient users who may come and go as they wish. Hence, there must be a decentralized protocol, executed by joining and leaving peers, that incrementally maintains the structure of the system. Additionally, a joining peer should be able to correctly execute this protocol while initially only having knowledge of a single, arbitrary participating network node.

One of the first overlay network projects was **Chord** [35], after which this encyclopedia entry is named (2001; Stoica, Morris, Karger, Kaashoek, Balakrishnan). More details about Chord are given below.

Key Results

The P2P area is very dynamic and rapidly evolving. The current entry provides a mere snapshot, covering dominant and characteristic strategies, but not offering an exhaustive survey.

Unstructured Overlays

Many of the currently deployed widespread resource-sharing networks have little or no particular overlay structure. More specifically, early systems such as Gnutella version 0.4 had no overlay structure at all, and allowed every node to connect to other nodes arbitrarily. This resulted in severe load and congestion problems.

Two-tier networks were introduced to reduce communication overhead and solve the scalability issues that early networks like Gnutella version 0.4 had. Two-tier networks consist of one tier of relatively stable and powerful nodes, called servers (superpeers, ultrapeers), and a larger tier of clients that search the network though servers. Most current networks, including Edonkey/Emule, KaZaa, and Gnutella, are built using two tiers. Servers provide directory store and search facilities. Searching is either limited to servers to which clients directly connect (eDonkey/eMule) or done by limited-depth flooding among the servers (Gnutella). The two-tier design considerably enhances the scalability and reliability of P2P networks. Nevertheless, the connections among servers and between clients/servers is done in a completely ad hoc manner. Thus, these networks provide no guarantee for the success of searches, nor a bound on their costs.

Structured Overlays Without Locality Awareness

Chord The Chord system was built at MIT and is currently being developed under FNSF's **IRIS** project (http://project-iris.net/). Several aspects of the Chord [35] design have influenced subsequent systems. We briefly explain the core structure of Chord here. Nodes have binary identifiers, assigned uniformly at random. Nodes are arranged in a linked ring according to their virtual identifiers. In addition, each node has shortcut links to other nodes along the ring, link i to a node 2^i away in the virtual identifier space. In this way, one can move gradually to the target by decreasing the distance by half at every step. Routing takes on average $\log n$ hops to reach any target, in a network containing n nodes. Each node maintains approximately $\log n$ links, providing the ability to route to geometrically increasing distances.

Constant Per-Node State Several overlay network algorithms were developed with the goal of pushing the amount of network state kept by each node in the overlay to a minimum. We refer to the state kept by a node as its *degree*, as it mostly reflects the number of connections to other nodes. **Viceroy** [23] was the first to demonstrate a dynamic network in which each node stores only five links to other network nodes, and routes to any other node in a logarithmic number of hops, $\log n$ for a network of n nodes. Viceroy provided a dynamic emulation of a butterfly network (see [11] for a textbook exposition of interconnect networks like butterfly). Later, several emulations of De Bruijn networks emerged, including the generic one of Abraham et al. (AAABMP) [1], the **distance halving** network [26], **D2B** [13], and **Koorde** [20]. Constant-degree overlay networks are too fragile for practical purposes, and may easily degrade in performance or even partition in the face of failures. A study of overlay networks under churn demonstrated these points [18]. Indeed, to the best of our knowledge, none of these constant-degree networks were built. Their main contribution, and the main reason for mentioning these works here, is to know that it is possible in principle to bring the per-node state to a bare, small constant.

Content Addressable Network The Content Addressable Network (CAN) [31] developed at ICSI builds the network as virtual d-dimensional space, giving every node a d-dimensional identifier. The routing topology resembles a d-dimensional torus. Routing is done by following the Euclidean coordinates in every dimension, yielding a $dn^{1/d}$ hop routing strategy. The parameter d can be tuned by the network administrator. Note that for $d = \log n$, CAN's features are the same as in Chord, namely, logarithmic degree and logarithmic routing hop count.

Overlay Routing Inspired by "Small-World" Networks The **Symphony** [24] algorithm emulates routing in a small world. Nodes have k links to nodes whose virtual identifiers are chosen at random according to a routable small-world distribution [22]. With k links, Symphony is expected to find a target in $\log^2 /k$ hops.

Overlay Networks Supporting Range Queries One of the deficiencies of DHTs is that they support only exact key lookup; hence, they do not address well the need to locate a range of keys, or to have a fuzzy search, e. g., search for any key that matches some prefix. **SkipGraphs** [4] and the **SkipNet** [19] scheme from Microsoft (project Herald) independently developed a similar DHT based on a ran-

P2P, Table 1
Comparison of various measures of lookup schemes with no locality awareness

Overlay lookup scheme	Topology resemblance	Hops	Degree
Chord	Hypercube	$\log n$	$\log n$
Viceroy	Butterfly	$\log n$	5
AAABMP, Distance-halving, Koorde, D2B	De Bruijn	$\log n$	4
Symphony	Small world	$\log^2 n/k$	k
SkipGraphs/SkipNet	Skip list	$\log n$	$\log n$
CAN	Torus	$dn^{1/d}$	d

domized skip list [28] that supports range queries over a distributed network. The approach in both of these networks is to link objects into a double-linked list, sorted by object names, over which "shortcut" pointers are built. Pointers from each object skip to a geometric sequence of distances in the sorted list, i. e., the first pointer jumps two items away, the second four items, and so on, up to pointer $\log n - 1$, which jumps over half of the list. Logarithmic, load-balanced lookup is achieved in this scheme in the same manner as in Chord. Because the identifier space is sorted by object names, rather than hash identifiers, ranges of objects can be scanned efficiently simply by routing to the lowest value in the range; the remaining range nodes reside contiguously along the ring.

By prefixing organization names to object names, SkipNet achieves contiguity of nodes belonging to a single organization along the ring, and the ability to map objects on nodes in their local organizations. In this way SkipNet achieves resource proximity and isolation the only system besides RP [33] to have this feature.

Whereas the SkipGraphs work focuses on randomized load-balancing strategies and proofs, the SkipNet system considers issues of dynamic maintenance, variable base sizes, and adopts the locality-awareness strategy of Pastry [33], which is described below.

Summary of Non-Locality-Aware Networks Each of the networks mentioned above is distinct in one or more of the following properties: The (intuitive) emulated **topology**; the expected number of **hops** required to reach a target; and the per-node **degree**. Table 1 summarizes these properties.

Locality Awareness

The problem with the approaches listed above is that they ignore the proximity of nodes in the underlying networks, and allow hopping back and forth across large physical distances in search of content. Recent studies of scalable content exchange networks [17] have indicated that up to 80% of Internet searches could be satisfied by local hosts within one's own organization. Therefore, even one far hop might be too costly. The next systems we encounter consider proximity relations among nodes in order to obtain locality awareness, i. e., that lookup costs are proportional to the actual distance of interacting parties.

Growth-Bounded Networks Several locality-aware lookup networks were built around a bit-fixing protocol that borrows from the seminal work of Plaxton et al. [27] (PRR). The *growth bounded* network model for which this scheme is aimed views the network as a metric space, and assumes that the densities of nodes in different parts of the network are not terribly different. The PRR [27] lookup scheme uses prefix routing, similar to Chord. It differs from Chord in that a link for flipping the ith identifier bit connects with any node whose length-i prefix matches the next hop. In this way, the scheme favors the closest one in the network. This strategy builds *geometric routing*, whose characteristic is that the routing steps toward a target increase geometrically in distance. This is achieved by having large flexibility in the choice of links for each prefix at the beginning of a route, and narrowing it down as the route progresses. The result is an overlay routing scheme that finds any target with a cost that is proportional to the shortest-distance route.

The systems that adopt the PRR algorithm are **Pastry** [33], **Tapestry** [36], and **Bamboo** [32]. A very close variant is **Kademlia** [25], in which links are symmetric. It is worth mentioning that the LAND scheme [2] improves PRR in providing a nearly optimal guaranteed locality guarantee; however, LAND has not been deployed.

Applications

Caching

The Coral network [14] from NYU, built on top of DSHT [15], has been operational since around 2004. It provides free content delivery services on top of the PlanetLab-distributed test bed [9], similar to the commercial services offered by the Akamai network. People use it to provide multiple, fast access points to content they wish to publish on the Web.

Coral optimizes access locality and download rate using locality-aware lookup provided by DSHT. Within Coral, DSHT is utilized to support locality-aware object location in two applications. First, Coral contains a collection of HTTP proxies that serve as content providers;

DSHT is used by clients for locating a close-by proxy. Second, proxy servers themselves use DSHT to locate a nearby copy of content requested by the client, thus making use of copies of the content that are stored in the network, rather than going to the source of the content.

Multicast

Several works deploy an event notification or publish–subscribe service over an existing routing overlay by building reverse-routing multicast paths from a single "target" to all "sources." For example, multicast systems built in this way include the Bayeux network [38], which is built over Tapestry [36], and SCRIBE [5], which is built over Pastry. In order to publish a file, the source advertises using flooding a tuple which contains the semantic name of a multicast session and a unique ID. This tuple is hashed to obtain a node identifier which becomes the session root node. Each node can join this multicast session by sending a message to the root. Nodes along the way maintain membership information, so that a multicast tree is formed in the reverse direction. The file content (and any updates) is flooded down the tree. Narada [8] is built with the same general architecture, but differs in its choice of links, and the maintenance of data.

Routing Infrastructure

A DHT can serve well to store routing and (potentially dynamic) location information of virtual host names. This idea has been utilized in a number of projects. A naming system for the Internet called CoDoNS [30] was built at Cornell University over the BeeHive overlay [29]. CoDoNS provides a safety net and is a possible replacement for the Domain Name System, the current service for looking up host names. Support for virtual IPv6 network addresses is provided in [37] by mapping names to their up-to-date, reachable IPv4 address. The Internet Indirection Infrastructure [34] built at the University of California, Berkeley provides support for virtual Internet host addresses that allows mobility.

Collaborative Content Delivery

Recent advances provide collaborative content delivery solutions that address both load balance and resilience via striping. The content is split into pieces (quite possibly with some redundancy through error-correcting codes). The source pushes the pieces of the file to an initial group of nodes, each of which becomes a source of a distribution tree for its piece, and pushes it to all other nodes. These works demonstrate clearly the advantages of data striping, i. e., of simultaneously exchanging stripes of data, over a tree-based dissemination of the full content.

SplitStream [6] employs the Pastry routing overlay in order to construct multiple trees, such that each participating node is an inner node in only one tree. It then supports parallel download of stripes within all trees. SplitStream [6] strives to obtain load balancing between multicast nodes. It achieves that by splitting the published content into several parts, called stripes, and publishing each part separately. Each stripe is published using a tree-based multicast. The workload is divided between the participating nodes by sending each stripe using a different multicast tree. Load balance is achieved by carefully choosing the multicast trees so that each node serves as an interior node in at most one tree. This reduces the number of "free riders" who only receive data.

A very popular file-distribution network is the BitTorrent system [10]. Nodes in BitTorrent are divided into *seed* nodes and *clients*. Seed nodes contain the desired content in full (either by being original providers, or by having completed a recent download of the content). Client nodes connect with a seed node or several seed nodes, as well as a *tracker* node, whose goal is to keep track of currently downloading clients. Each client selects a group (currently, of size about 20) of other downloading clients, and exchanges chunks of data obtained from the seed(s) with them. BitTorrent employs several intricate strategies for selecting which chunks to request from what other clients, in order to obtain fair load sharing of the content distribution and, at the same time, achieve fast download.

BitTorrent currently does not contain P2P-searching facilities. It relies on central sites known as "trackers" to locate content, and to coordinate the BitTorrent download process. Recent announcements by Bram Cohen (the creator of BitTorrent) and creators of other BitTorrent clients state that new protocols based on BitTorrent will be available soon, in which the role of trackers is eliminated, and searching and coordination is done in a completely P2P manner.

Experience with BitTorrent and similar systems indicates that the main problem with this approach is that towards the end of a download, many peers may be missing the same rare chunks, and the download slows down. Fairly sophisticated approaches were published in an attempt to overcome this issue.

Recently, a number of works at Microsoft Research have demonstrated the benefits of network coding in efficient multicast, e. g., [7] and Avalanche [16]. We do not cover these techniques in detail here, but only briefly state the principal ideas that underlie them.

The basic approach in network coding is to re-encode all the chunks belonging to the file, so that each one that is shared is actually a linear combination of all the pieces. The blocks are then distributed with a description of the content. Once a node obtains these re-encoded chunks, it can generate new combinations from the ones it has, and can send those out to other peers. The main benefit is that peers can make use of any new piece, instead of having to wait for specific chunks that are missing. This means no one peer can become a bottleneck, since no piece is more important than any other. Once a peer collects sufficiently many such chunks, it may use them to reconstruct the whole file.

It is worth noting that in unstructured settings, it was recently shown that network coding offers no advantage [12].

Cross References

▶ Geometric Spanners
▶ Routing
▶ Sparse Graph Spanners

Recommended Reading

1. Abraham, I., Awerbuch, B., Azar, Y., Bartal, Y., Malkhi, D., Pavlov, E.: A generic scheme for building overlay networks in adversarial scenarios. In: Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS 2003), 2003
2. Abraham, I., Malkhi, D., Dobzinski, O.: LAND: Stretch $(1+\varepsilon)$ locality aware networks for DHTs. In: Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA04), 2004
3. Abraham, I., Badola, A., Bickson, D., Malkhi, D., Maloo, S., Ron, S.: Practical locality-awareness for large scale information sharing. In: The 4th Annual International Workshop on Peer-To-Peer Systems (IPTPS '05), 2005
4. Aspnes, J., Shah, G.: Skip graphs. In: Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, Baltimore, January 2003, pp. 384–393
5. Castro, M., Druschel, P., Rowstron, A.: Scribe: A large-scale and decentralised application-level multicast infrastructure, IEEE J. Sel. Areas Commun. (JSAC) (Special issue on Network Support for Multicast Communications) **20**(8), 1489–1499 (2002). ISSN: 0733–8716
6. Castro, M., Druschel, P., Kermarrec, A.-M., Nandi, A., Rowstron, A., Singh, A.: Splitstream: High-bandwidth multicast in a cooperative environment. In: SOSP'03, October 2003
7. Chou, P., Wu, Y., Jain, K.: Network coding for the internet. In: IEEE Communication Theory Workshop, 2004
8. Chu, Y., Rao, S.G., Zhang, H.: A case for end system multicast. In: Proceedings of ACM SIGMETRICS, Santa Clara, June 2000, pp. 1–12
9. Chun, B., Culler, D., Roscoe, T., Bavier, A., Peterson, L., Wawrzoniak, M., Bowman, M.: Planetlab: An overlay testbed for broad-coverage services. ACM SIGCOMM Comput. Commun. Rev. **33**, 3–12 (2003)
10. Cohen, B.: Incentives build robustness in bittorrent. In: Proceedings of P2P Economics Workshop, 2003
11. Cormen, T.H., Leiserson, C.E., Rivest, R.L.: Introduction to algorithms. MIT Press (1990)
12. Fernandess, Y., Malkhi, D.: On collaborative content distribution using multi-message gossip. In: Twentieth IEEE International Parallel and Distributed Processing Symposium (IPDPS 2006), Greece, April 2006
13. Fraigniaud, P., Gauron, P.: The content-addressable network D2B. Tech. Report 1349, LRI, Univ. Paris-Sud (2003)
14. Freedman, M.J., Freudenthal, E., Mazières, D.: Democratizing content publication with coral. In: Proceedings of the 1st USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI '04), March 2004
15. Freedman, M.J., Mazières, D.: Sloppy hashing and self-organizing clusters. In: Proceedings of the 2nd Intl. Workshop on Peer-to-Peer Systems (IPTPS '03), February 2003
16. Gkantsidis, C., Rodriguez, P.: Network coding for large scale content distribution. In: IEEE/INFOCOM, 2005
17. Gummadi, K.P., Dunn, R.J., Saroiu, S., Gribble, S.D., Levy, H.M., Zahorjan, J.: Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. In: Proceedings of the nineteenth ACM symposium on Operating systems principles, pp. 314–329. ACM Press (2003)
18. Gummadi, K., Gummadi, R., Gribble, S., Ratnasamy, S., Shenker, S., Stoica, I.: The impact of DHT routing geometry on resilience and proximity. In: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications, pp. 381–394. ACM Press (2003)
19. Harvey, N.J.A., Jones, M.B., Saroiu, S., Theimer, M., Wolman, A.: Skipnet: A scalable overlay network with practical locality properties. In: Proceedings of Fourth USENIX Symposium on Internet Technologies and Systems (USITS '03), March 2003
20. Kaashoek, F., Karger, D.R.: Koorde: A simple degree-optimal hash table. In: 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03), 2003
21. Karger, D., Lehman, E., Leighton, F.T., Levine, M., Lewin, D., Panigrahy, R.: Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In: Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC), 1997, pp. 654–663 1997
22. Kleinberg, J.: The small-world phenomenon: An algorithmic perspective. In: Proc. 32nd ACM Symposium on Theory of Computing (STOC 2000), 2000, pp. 163–170
23. Malkhi, D., Naor, M., Ratajczak, D.: Viceroy: A scalable and dynamic emulation of the butterfly. In: Proceedings of the 21st ACM Symposium on Principles of Distributed Computing (PODC '02), 2002, pp. 183–192
24. Manku, G.S., Bawa, M., Raghavan, P.: Symphony: Distributed hashing in a small world. In: Proc. 4th USENIX Symposium on Internet Technologies and Systems (USITS 2003) 2003, pp. 127–140
25. Maymounkov, P., Mazières, D.: Kademlia: A peer-to-peer information system based on the XOR metric. In: Proc. 1st Intl. Workshop on Peer-to-Peer Systems (IPTPS 2002), 2002, pp. 53–65
26. Naor, M., Wieder, U.: Novel architectures for p2p applications: the continuous-discrete approach. In: The Fifteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '03), 2003
27. Plaxton, C., Rajaraman, R., Richa, A.: Accessing nearby copies of replicated objects in a distributed environment. In: Proceedings of the Ninth Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA 97), 1997, pp. 311–320

28. Pugh, W.: Skip lists: A probabilistic alternative to balanced trees. In: Workshop on Algorithms and Data Structures, 1989, pp. 437–449
29. Ramasubramanian, V., Sirer, E.G.: Beehive: O(1) lookup performance for power-law query distributions in peer-to-peer overlays. In: Proceedings of Networked System Design and Implementation (NSDI), 2004
30. Ramasubramanian, V., Sirer, E.G.: The design and implementation of a next generation name service for the internet. In: Proceedings of SIGCOMM, 2004
31. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A scalable content-addressable network. In: Proceedings of the ACM SIGCOMM 2001 Technical Conference, 2001
32. Rhea, S., Geels, D., Roscoe, T., Kubiatowicz, J.: Handling churn in a dht. Tech. Report Technical Report UCB//CSD-03-1299, The University of California, Berkeley, December 2003
33. Rowstron, A., Druschel, P.: Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In: IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), 2001, pp. 329–350
34. Stoica, I., Adkins, D., Zhuang, S., Shenker, S., Surana, S.: Internet Indirection Infrastructure. In: Proceedings of ACM SIGCOMM, pp. 73–88 (2002)
35. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. In: Proceedings of the SIGCOMM 2001
36. Zhao, B.Y., Huang, L., Stribling, J., Rhea, S.C., Joseph, A.D., Kubiatowicz, J.: Tapestry: A resilient global-scale overlay for service deployment. IEEE J. Sel. Areas Commun. (2003)
37. Zhou, L., van Renesse, R., Marsh, M.: Implementing IPv6 as a Peer-to-Peer Overlay Network. In: Proceedings of the 21st IEEE Symposium on Reliable Distributed Systems (SRDS'02), pp. 347 (2002)
38. Zhuang, S.Q., Zhao, B.Y., Joseph, A.D., Katz, R.H., Kubiatowicz, J.: Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination. In: Proceedings of the Eleventh International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV 2001), 2001

Packet Routing

1988; Leighton, Maggs, Rao

LENORE J. COWEN
Department of Computer Science, Tufts University, Medford, MA, USA

Keywords and Synonyms

Store-and-forward routing; Job shop scheduling

Problem Definition

A collection of packets need to be routed from a set of specified sources to a set of specified destinations in an arbitrary network. Leighton, Maggs and Rao [5] looked at a model where this task is divided into two separate tasks: the first is the *path selection* task, where for each specified packet i with source s_i and packet destination t_i, a simple (meaning edges don't repeat) path P_i through the network from s_i to t_i is pre-selected. Packets traverse the network in a *store and forward* manner: each time a packet is forwarded it travels along the next link in the pre-selected path. It is assumed that only one packet can cross each individual link at each given global (synchronous) timestep. Thus, when there is contention for a link, packets awaiting traversal are stored in the local link's queue (special source and sink queues of unbounded size are also defined that store packets at their origins and destinations). Thus, the second task, and the focus of the Leighton, Maggs and Rao result (henceforth called the LMR result) is the *scheduling* task: a determination, when a link's queue is not empty, of which packet gets to traverse the link in the next timestep (where it is assumed to immediately join the link queue for its next hop). The goal is to schedule the packets so that the *maximum* time that it takes any packet to reach its destination is minimized.

There are two parameters of the network together with the pre-selected paths that are clearly relevant. One is the *congestion* c, defined as the maximum number of paths that all use the same link. The other is the *dilation* d, which is simply the length of the longest path that any packet traverses in the network. Clearly each of c and d is a lower-bound on the length of any schedule that routes all the packets to their destinations. It is easy to see that a schedule of length at most cd always exists. In fact, any schedule that never lets a link go idle if there is a packet that can use that link at that timestep is guaranteed to terminate in cd steps, because each packet traverses at most d links, and at any link can be delayed by at most $c - 1$ other packets.

Key Results

The surprising and beautiful result of LMR is as follows:

Theorem ([5]) *For any network G with a pre-specified set of paths P with congestion c and dilation d, there exists a schedule of length $O(c + d)$, where the queue sizes at each edge are always bounded by a constant.*

The original proof of the LMR paper is non-constructive. That is, it uses the Local Lemma [3] to prove the existence of such a schedule, but does not give a way to find it. In his book [10], Scheideler showed that in fact, a $O(c + d)$ schedule exists with edge queue sizes bounded by 2 (and gave a simpler proof of the original LMR result). A subsequent paper of Leighton, Maggs and Richa in 1999 [6] provides a constructive version of the original LMR paper as follows:

Theorem ([6]) *For any network G with a pre-specified set of paths P with congestion c and dilation d, there exists* *a schedule of length $O(c + d)$. Furthermore, such a schedule can be found in $O(p \log^{1+\epsilon} p \log^*(c+d))$ time for any $\epsilon>0$, where p is the sum of the lengths of the paths taken by the packets and ε is incorporated into the constant hidden by the big-O in the schedule length.*

The algorithm in the paper is a randomized one, though the authors claim that it can be derandomized using the method of conditional probabilities. However, even though the algorithm of Leighton, Maggs and Richa is constructive, it is still an offline algorithm: namely, it requires full knowledge of all packets in the network and the precise paths that each will traverse in order to construct the schedule. The original LMR paper also gave a simple randomized *online* algorithm, that, by assigning delays to packets independently and uniformly at random from an appropriate interval, results in a schedule which is much better than greedy schedules, though not as good as the offline constructions.

Theorem ([5]) *There is a simple randomized on-line algorithm for producing, with high probability, a schedule of length $O(c + d\log(Nd))$ using queues of size $O(\log(Nd))$, where c is the congestion, d is the dilation, and N is the number of packets.*

In the special case where it is assumed that all packets follow shortest paths in the network, Meyer, auf der Heide and Vöcking produced a simple randomized online algorithm that produces, with high probability, a schedule of length $O(c + d + \log Nd)$ steps, but queues can be as large as $O(c)$ [7]. For arbitrary paths, the LMR online result was ultimately improved to $O(c + d + \log^{1+\epsilon} N)$ steps, for any $\epsilon>0$ with high probability, in a series of two papers by Rabani and Tardos [9], and Rabani and Ostrovsky [8]. Online protocols have also been studied in a setting where additional packets are dynamically injected into the network in adversarial settings, see [10] for a survey.

The discussion is briefly returned to the first task, namely to pre-construct the set of paths. Clearly, the goal is to find, for a particular set of packets with pre-specified sources and destinations, a set of paths that minimizes $c + d$. Srinivasan and Teo [12] designed an off-line algorithm that produces a set of paths whose $c + d$ is provably within a constant factor of optimal. Together with the offline LMR result, that gives a constant-factor approximation problem for the offline store-and-forward packet routing problem. Note that the approach of trying to minimize $c + d$ rather than c alone seems crucial; producing schedules within a constant factor of optimal congestion c is hard, and in fact has been shown to be related to the integrality gap for multicommodity flow [1,2].

Applications

Network Emulations

Typically, a guest network G is emulated by a host network H by embedding G into H. Nodes of G are mapped to nodes of H, while edges of G are mapped to paths in H. If P is the set of e paths (each corresponding to an edge in the guest network G), the congestion and dilation can be defined analogously as in the main result for the set of paths P, namely c denotes the maximum number of paths that use any one edge of H, and d is the length of the longest path in P. In addition, the *load* l is defined to be the maximum number of nodes in G that are mapped to a single node of H. Once G is embedded in H, H can emulate G as follows: Each node of H emulates the local computations performed by the l (or fewer) nodes mapped to it in $O(l)$ time. Then for each packet sent along an edge of G, H sends a packet along the corresponding path in the embedding; using the offline LMR result this takes $O(c+d)$ steps. Thus, H can emulate each step of G in $O(c + d + l)$ steps.

Job Shop Scheduling

Consider a scheduling problem with jobs $j_1, \ldots j_r$ and machines $m_1, \ldots, m_s$ for which each job must be performed on a specified sequence of machines (in a specified order). Assume each job spends unit time on each machine, and that no machine has to work on any job more than once (In the language of job-shop scheduling, this is the *non-preemptive, acyclic, job-shop scheduling problem*, with unit jobs). There is a mapping of sequences of machines to paths and jobs to packets so that this becomes an encoding of the main packet routing problem, where if c is now to be the maximum number of jobs that have to be run on any one machine, and d to be the maximum number of different machines that work on any single job, there becomes $O(c)$ congestion and $O(d)$ dilation for the corresponding packet-routing instance. Then the offline LMR result shows that there is a schedule that completes all jobs in $O(c+d)$ steps, where in addition, each job waits at most a constant number of steps in between consecutive machines (and the queue of jobs waiting for any particular machine will always be bounded by a constant). Similar techniques to those developed in the LMR paper have subsequently been applied to more general instances of Job-Shop Scheduling; see [4,11].

Open Problems

The main open problem is whether there is a randomized *online* packet scheduling that matches the offline LMR bound of $O(c+d)$. The bound of [8] is close, but still grows logarithmically with the total number of packets.

For job shop scheduling, it is unknown whether the constant-factor approximation algorithm for the non-preemptive acyclic job-shop scheduling problem with unit length jobs implied by LMR can be improved to a PTAS. It is also unknown whether there is a constant-factor approximation in the case of arbitrary-length jobs.

Recommended Reading

1. Andrews, M., Zhang, L.: Hardness of the Undirected Congestion Minimization Problem. Proceedings of the 37th Annual ACM Symposium on Theory of Computing, pp. 284–293 (2005)
2. Chuzhoy, J., Naor, J.: New Hardness Results for Congestion Minimization and Machine Scheduling. Proceedings of the 36th Annual ACM Symposium on Theory of Computing, pp. 28–34. ACM, New York (2004)
3. Erdös, P., Lovász, L.: Problems and results on 3-chromatic hypergraphs and some related questions. Colloq. Math. Soc. János Bolyai **10**, 609–627 (1975)
4. Goldberg, L.A., Patterson, M., Srinivasan, A., Sweedick, E.: Better Approximation Guarantees for Job-Shop Scheduling. SIAM J. Discret. Math. **14**(1), 67–92 (2001)
5. Leighton, F.T., Maggs, B.M., Rao, S.B.: Packet routing and jobshop scheduling in O(congestion+dilation) steps. Combinatorica **14**(2), 167–180 (1994)
6. Leighton, F.T., Maggs, B.M., Richa, A.W.: Fast algorithms for finding O(congestion+dilation) packet routing schedules. Combinatorica **19**(3), 375–401 (1999)
7. Meyer auf der Heide, F., Vöcking, B.: Shortest-Path Routing in Arbitrary Networks. J. Algorithms **31**(1), 105–131 (1999)
8. Ostrovsky, R., Rabani, Y.: Universal $O(\text{congestion} + \text{dilation} + \log^{1+\varepsilon} N)$ Local Control Packet Switching Algorithm. In: Proceedings of The Twenty-Ninth ACM Symposium on Theory of Computing, pp. 644–653 (1997)
9. Rabani, Y., Tardos, E.: Distributed Packet Switching in Arbitrary Networks. In: the 28th ACM Symposium on Theory of Computing, pp. 366–376 (1996)
10. Scheideler, C.: Universal Routing Strategies for Interconnection Networks. In: Lecture Notes in Computer Science, vol. 1390. Springer (1998)
11. Shmoys, D.B., Stein, C., Wein, J.: Improved Approximation Algorithms for Shop Scheduling Problems. SIAM J. Comput. **23**(3), 617–632 (1994)
12. Srinivasan, A., Teo, C.P.: A Constant-Factor Approximation Algorithm for Packet Routing and Balancing Local vs. Global Criteria. SIAM J. Comput. **30**(6), 2051–2068 (2000)

Packet Switching in Multi-Queue Switches

2004; Azar, Richter; Albers, Schmidt

MARKUS SCHMIDT
Institute for Computer Science, University of Freiburg, Freiburg, Germany

Keywords and Synonyms

Online packet buffering; Online packet routing

Problem Definition

A multi-queue network switch serves m incoming queues by transmitting data packets arriving at m input ports through one single output port. In each time step, an arbitrary number of packets may arrive at the input ports, but only one packet can be passed through the common output port. Each packet is marked with a value indicating its priority in the Quality of Service (QoS) network. Since each queue has bounded capacity B and the rate of arriving packets can be much higher than the transmission rate, packets can be lost due to insufficient queue space. The goal is to maximize the throughput which is defined as the total value of transmitted packets. The problem comprises two dependent questions: buffer management, namely which packets to admit into the queues, and scheduling, i. e. which (FIFO) queue to use for transmission in each time step.

Two scenarios are distinguished: (a) unit packet value (All packets have the same value.), (b) arbitrary packet values.

The problem is considered as an online problem, i. e. at time step t, only the packet arrivals until t are known, but nothing about future packet arrivals. The online switch performance in QoS based networks is studied by using competitive analysis in which the throughput of the online algorithm is compared to the throughput of an optimal offline algorithm knowing the whole arrival sequence in advance.

If not stated otherwise, the admission control is assumed to allows preemption, i. e. packets once enqueued need not necessarily be transmitted, but can be discarded.

Problem 1 (Unit value problem) *All packets have value 1. Since all packets are thus equally important, the admission control policies simplify: All arriving packets are to be enqueued; in the case of buffer overflow, it does not matter which packets are stored in the queue and which packets are discarded.*

Problem 2 (General problem) *Each packet has its individual value where usually a range $[1, \alpha]$ is given for all packets. A special case consists in the two value model where the values are restricted to $\{1, \alpha\}$.*

Key Results

Unit value packets

Deterministic algorithms

Theorem 1 ([1]) *For any buffer size B, the competitive ratio of each deterministic online algorithm is not smaller than $(e_B + \frac{2}{B})/(e_B - 1 + \frac{1}{B}) \geq \frac{e}{e-1} \approx 1.58$ where $e_B = ((B+1)/B)^B$.*

Theorem 2 ([4]) *Every work-conserving online algorithm is 2-competitive.*

Theorem 3 ([1]) *For any buffer size B, the competitive ratio of any greedy algorithm, which always serves a longest queue (LQF), is at least $2 - \frac{1}{B}$ if $m \gg B$.*

Algorithm: *SGR* (Semi-Greedy)
In each time step, the algorithm executes the first rule that applies to the current buffer configuration.

1. If there is a queue buffering more than $\lfloor B/2 \rfloor$ packets, serve the queue currently having the maximum load.
2. If there is a queue the hitherto maximum load of which is less than B, serve among these queues the one currently having the maximum load.
3. Serve the queue currently having the maximum load.

Ties are broken by choosing the queue with the smallest index. The hitherto maximum load is reset to 0 for all queues whenever all queues are unpopulated in *SGR*'s configuration.

Theorem 4 ([1]) *If B is even, then SGR is $\frac{17}{9} \approx 1.89$-competitive. If B is odd, then SGR is $(\frac{17}{9} + \frac{\delta_B}{9})$-competitive where $\delta_B = \frac{2}{B+1}$.*

Theorem 5 ([3]) *Algorithm $E^{M^{\hat{EP}'}}$ (not stated in detail due to space limitation), which is based on a water level algorithm and uses a fractional matching in an online constructed graph, achieves a competitiveness of $e/(e-1)(1 + (\lfloor H_m + 1 \rfloor)/B)$, where H_m denotes the m^{th} harmonic number. Thus, $E^{M^{\hat{EP}'}}$ is asymptotically $\frac{e}{e-1}$-competitive for $B \gg \log m$.*

Randomized algorithms

Theorem 6 ([1]) *The competitive ratio of each randomized online algorithm is at least $\varrho = 1.4659$ for any buffer size B ($\varrho = 1 + \frac{1}{\alpha+1}$ where α is the unique positive root of $e^{\alpha} = \alpha + 2$).*

Theorem 7 (Generalizing technique [9]**)** *If there is a randomized c-competitive algorithm A for B = 1, then there is a randomized c-competitive algorithm $\tilde{A}$ for all B.*

Algorithm: *RS* (Random Schedule)

1. The algorithm uses m auxiliary queues $Q_1, \ldots, Q_m$ of sizes $B_1, \ldots, B_m$ (different buffer sizes at the distinct ports are allowed), respectively. These queues contain real numbers from the range (0,1), where each number is labeled as either marked or unmarked. Initially, these queues are empty.
2. Packet arrival: If a new packet arrives at queue q_i, then the algorithm chooses uniformly at random a real number from the range (0,1) that is inserted into queue Q_i and labeled as unmarked. If queue Q_i was full when the packet arrived, the number at the head of the queue is deleted prior to the insertion of the new number.
3. Packet transmission: Check whether queues $Q_1, \ldots, Q_m$ contain any unmarked number. If there are unmarked numbers, let Q_i be the queue containing the largest unmarked number. Change the label of the largest number to "marked" and select queue q_i for transmission. Otherwise (no unmarked number), transmit a packet from any non-empty queue if such exists.

Theorem 8 ([4]) *Randomized algorithm RS is $\frac{e}{e-1} \approx 1.58$-competitive.*

Algorithm: *RP* (Random Permutation)
Let $\mathcal{P}$ be the set of permutations of $\{1, \ldots, m\}$, denoted as m-tuples. Choose $\pi \in \mathcal{P}$ according to the uniform distribution and fix it. In each transmission step, choose among the populated queues that one whose index is most to the front in the m-tuple π.

Theorem 9 ([9]) *Randomized algorithm RP is $\frac{3}{2}$-competitive for B = 1. By Theorem 7, there is a randomized algorithm $\tilde{RP}$ that is $\frac{3}{2}$-competitive for arbitrary B.*

Arbitrary value packets

Definition 1 A switching algorithm *ALG* is called *comparison-based* if it bases its decisions on the relative order between packet values (by performing only comparisons), with no regard to the actual values.

Theorem 10 (Zero-one principle [5]**)** *Let ALG be a comparison-based switching algorithm (deterministic or randomized). ALG is c-competitive if and only if ALG achieves a c-competitiveness for all packet sequences whose values are restricted to $\{0, 1\}$ for every possible way of breaking ties between equal values.*

Algorithm: *GR* (Greedy)
Enqueue a new packet if

- the queue is not full
- or a packet with the smallest value in the queue has a lower value then the new packet. In this case, a smallest value packet is discarded and the new packet in enqueued.

Algorithm: *TLH* (Transmit Largest Head)

1. Buffer management: Use algorithm *GR* independently in all m incoming queues.

2. Scheduling: At each time step, transmit the packet with the largest value among all packets at the head of the queues.

Theorem 11 ([5]) *Algorithm TLH is 3-competitive.*

Algorithm: *TL* (Transmit Largest)
1. Buffer management: Use algorithm *GR* independently in all m incoming queues.
2. Scheduling: At each time step, transmit the packet with the largest value among all packets stored in the queues.

Algorithm: GS^A (Generic Switch)
1. Buffer management: Apply buffer management policy A to all m incoming queues.
2. Scheduling: Run a simulation of algorithm *TL* (in the preemptive relaxed model) with the online input sequence σ. Adopt all scheduling decisions of *TL*, i. e. at each time step, transmit the packet at the head of the queue used by *TL* simulation.

Theorem 12 (General reduction [4]**)** *Let GS^A denote the algorithm obtained by running algorithm GS with the event-driven single-queue buffer management policy A (preemptive or non-preemptive) and let c_A be the competitive ratio of A. The competitive ratio of GS^A satisfies $c_{GS^A} \leq 2 \cdot c_A$.*

Applications

The unit value scenario models most current networks, e. g. IP networks which only support a "best effort" service in which all packet streams are treated equally, whereas the scenario with arbitrary packet values integrates full QoS capabilities.

The general reduction technique allows to restrict oneself to investigate single-queue buffer problems. It can be applied to a 1.75-competitive algorithm named *PG* by Bansal et al. [7], which achieves the best ratio known today, and yields an algorithm GS^{PG} that is 3.5-competitive for multi-queue buffers (3.5 is still higher than 3 which is the competitive ratio of TLH). In the 2-value preemptive model, Lotker and Patt-Shamir [8] presented a *mark&flush* algorithm *mf* that is 1.30-competitive for single queue buffers and that the general reduction technique transforms into a 2.60-competitive algorithm GS^{mf} for multi-queue buffers.

For the general non-preemptive model, Andelman et al. [2] presented a policy for a single queue called *Exponential-Interval Round-Robin* (*EIRR*), which is $(e\lceil\ln\alpha\rceil)$-competitive, and showed also a lower bound of $\Theta(\log\alpha)$. In the multi-queue buffer case, the general reduction technique provides a non-preemptive $(e\lceil\ln\alpha\rceil)$-competitive algorithm.

Open Problems

It is known from Theorem 3 that the competitive ratio of any greedy algorithm in the unit value model is at least 2 if $m \gg B$. Which is the tight upper bound for greedy algorithms in the opposite case $B \gg m$?

The proof of the lower bound $e/(e-1)$ in Theorem 1 uses $m \gg B$ whereas Theorem 5 achieves $e/(e-1)$ as an upper bound for $B \gg \log m$. In [4], a lower bound of 1.366 is shown, independent of B and m. Which is the optimal competitive ratio for arbitrary B and m?

Due to the general reduction technique in Theorem 7, the competitive ratio for multi-queue buffer algorithms can be improved if better competitiveness results for single queue buffer algorithms are achieved. Currently, $\frac{\sqrt{13}+5}{6} \approx 1.43$ [2] and 1.75 [7] are the best known lower and upper bounds, respectively. How to reduce this gap?

Cross References

▶ Packet Switching in Single Buffer
▶ Paging

Recommended Reading

1. Albers, S., Schmidt, M.: On the performance of greedy algorithms in packet buffering. SIAM J. Comput. **35**, 278–304 (2005)
2. Andelman, N., Mansour, Y., Zhu, A.: Competitive queueing policies for QoS switches. In: Proc. 14th ACM-SIAM Symp. on Discrete Algorithms (SODA), 761–770 (2003)
3. Azar, Y., Litichevskey, M.: Maximizing throughput in multi-queue switches. In: Proc. 12th Annual European Symp. on Algorithms (ESA), 53–64 (2004)
4. Azar, Y., Richter, Y.: Management of multi-queue switches in QoS Networks. In: Proc. 35th ACM Symp. on Theory of Computing (STOC), 82–89 (2003)
5. Azar, Y., Richter, Y.: The zero-one principle for switching networks. In: Proc. 36th ACM Symp. on Theory of Computing (STOC), 64–71 (2004)
6. Azar, Y., Richter, Y.: An improved algorithm for CIOQ switches. In: Proc. 12th Annual European Symp. on Algorithms (ESA). LNCS, vol. 3221, 65–76 (2004)
7. Bansal, N., Fleischer, L., Kimbrel, T., Mahdian, M., Schieber, B., Sviridenko, M.: Further improvements in competitive guarantees for QoS buffering. In: Proc. 31st International Colloquium on Automata, Languages, and Programming (ICALP), 64–71 (2004)
8. Lotker, Z., Patt-Shamir, B.: Nearly optimal FIFO buffer management for two packet classes. Comput. Netw. **42**(4), 481–492 (2003)
9. Schmidt, M.: Packet buffering: randomization beats deterministic algorithms. In: Proc. 22nd Annual Symp. on Theoretical Aspects of Computer Science (STACS). LNCS, vol. 3404, 293–304 (2005)

Packet Switching in Single Buffer

2003; Bansal, Fleischer, Kimbrel, Mahdian, Schieber, Sviridenko

ROB VAN STEE
Algorithms and Complexity Department, Max Planck Institute for Computer Science, Saarbrücken, Germany

Keywords and Synonyms

Buffering

Problem Definition

In this entry, consider a Quality of Service (QoS) buffering system that is able to hold B packets. Time is slotted. At the beginning of a time step a set of packets (possibly empty) arrives and at the end of the time step a single packet may leave the buffer to be transmitted. Since the buffer has a bounded size, at some point packets may need to be dropped. The buffer management algorithm has to decide at each step which of the packets to drop and which packets to transmit, subject to the buffer capacity constraint. The value of a packet p is denoted by $v(p)$. The system obtains the value of the packets it sends, and gains no value otherwise. The aim of the buffer management algorithm is to maximize the total value of transmitted packets.

In the FIFO model, the packet transmitted at time t is always the first (oldest) packet in the buffer.

In the *nonpreemptive* model, packets accepted to the queue will be transmitted eventually and cannot be dropped. In this model, the best competitive ratio achievable is $\Theta(\log \alpha)$ where α is the ratio of the maximum value of a packet to the minimum [1,2].

In the *preemptive* model, packets can also be dropped at some later time before they are served. The rest of this entry focuses on this model. Mansour, Patt-Shamir, and Lapid [9] were the first to study preemptive queuing policies for a single FIFO buffer, proving that the natural greedy algorithm (see definition in Fig. 1) maintains a competitive ratio of at most 4. This bound was improved to the tight value of 2 by Kesselman, Lotker, Mansour, Patt-Shamir, Schieber, and Sviridenko [6].

The greedy algorithm is not optimal since it never preempts a packet until the buffer is full and this might be too late. The first algorithm with a competitive ratio strictly below 2 was presented by Kesselman, Mansour, and van Stee [7]. This algorithm uses a parameter β and introduces an extra rule for processing arrivals, that is executed before rules 1 and 2 of the greedy algorithm. This rule is formulated in Fig. 2.

The Greedy Algorithm.
When a packet of value $v(p)$ arrives:
1. Accept p if there is free space in the buffer.
2. Otherwise, reject (drop or preempt) the packet p' that has minimal value among p and the packets in the buffer. If $p' \neq p$, accept p.

Packet Switching in Single Buffer, Figure 1
The natural greedy algorithm

0. Preempt (drop) the first packet p' in the FIFO order such that $v(p') \leq v(p)/\beta$, if any (p *preempts* p').

Packet Switching in Single Buffer, Figure 2
Extra rule for the preemptive greedy algorithm

0'. Find the first (i. e., closest to the front of the buffer) packet p' such that p' has value less than $v(p)/\beta$ and not more than the value of the packet after p' in the buffer (if any). If such a packet exists, drop it (p *preempts* p').

Packet Switching in Single Buffer, Figure 3
Modified preemptive greedy

It is shown in [7] that by taking $\beta = 15$, the algorithm preemptive greedy (PG) has a competitive ratio of 1.983. The analysis is rather complicated and is done by assigning the value of packets served by the offline algorithm to packets served by PG.

A lower bound of 5/4 for this problem was shown in [9]. This was improved to $\sqrt{2}$ in [2] and then to 1.419 in [7].

Key Results

A modification of PG was presented by Bansal, Fleischer, Kimbrel, Mahdian, Schieber, and Sviridenko [3]. It changes rule 0 to rule 0'.

Thus, the modification compared to PG is that this algorithm finds a "locally optimal" packet to evict. We will denote modified preemptive greedy by MPG.

Theorem 1 ([3]) *For $\beta = 4$, MPG has a competitive ratio of 1.75.*

The proof begins by showing that in order to analyze the performance of MPG, it is sufficient to consider only input instances in which the value of each packet is either 0 or β^i for some $i \geq 0$, but ties are allowed to be broken by the adversary.

The authors then define an *interval structure* for input instances. An interval I is said to be of type i if at every step $t \in I$ MPG outputs a packet of value at least β^i, and I is a maximal interval with this property.

$\mathcal{I}_i$ is the collection of maximal intervals of type i, and $\mathcal{I}$ is the union of all $\mathcal{I}_i$'s. This is a multiset, since an interval of type i can also be contained in an interval of one or more types $j < i$.

This induces an interval structure which is a sequence of ordered rooted trees in a natural way: the root of each tree is an interval in $\mathcal{I}_0$, and the children of each interval $I \in \mathcal{I}_i$ are all the maximal intervals of type $i + 1$ which are contained in I. These children are ordered from left to right based on time, as are the trees themselves. The intervals of type i (and the vertices that represent them) are distinguished by whether or not an eviction of a packet of value at least β^i occurred during the interval.

To complete the proof, the authors show that for every interval structure $\mathcal{T}$, the competitive ratio of MPG on instances with interval structure $\mathcal{T}$ can be bounded by the solution of a linear program indexed by $\mathcal{T}$. Finally, it is shown that for every $\mathcal{T}$ and every $\beta \geq 4$, the solution of this program is at most $2 - 1/\beta$.

Applications

In recent years, there has been a lot of interest in Quality of Service networks. In regular IP networks, packets are indistinguishable and in case of overload any packet may be dropped. In a commercial environment, it is much more preferable to allow better service to higher-paying customers or customers with critical requirements. The idea of Quality of Service guarantees is that packets are marked with values which indicate their importance.

This naturally leads to decision problems at network switches when many packets arrive and overload occurs. The algorithm presented in this entry can be used to maximize network performance in a network which supports Quality of Service.

Open Problems

Despite substantial advances in improving the upper bound for this problem, a fairly large gap remains. Sgall [5] showed that the performance of PG is as good as that of MPG. Recently, Englert and Westermann [4] showed that PG has a competitive ratio of at most $\sqrt{3} \approx 1.732$ and at least $1 + 1/2\sqrt{2} \approx 1.707$. Thus, to improve further, a different algorithm will be needed.

The authors also note that Lotker and Patt-Shamir [8] studied the special case of two packet values and derived a 1.3-competitive algorithm, which closely matches the corresponding lower bound of 1.28 from Mansour et al. [9]. An open problem is to close the remaining small gap.

Cross References

► Packet Switching in Multi-Queue Switches

Recommended Reading

1. Aiello, W., Mansour, Y., Rajagopolan, S., Rosen, A.: Competitive queue policies for differentiated services. In: Proc. of the IEEE INFOCOM, pp. 431–440. IEEE, Tel-Aviv, Israel (2000)
2. Andelman, N., Mansour, Y., Zhu, A.: Competitive queueing policies in QoS switches. In: Proc. 14th Symp. on Discrete Algorithms (SODA), pp. 761–770 ACM/SIAM, San Francisco, CA, USA (2003)
3. Bansal, N., Fleischer, L., Kimbrel, T., Mahdian, M., Schieber, B., Sviridenko, M.: Further improvements in competitive guarantees for QoS buffering. In: Proc. 31st International Colloquium on Automata, Languages, and Programming (ICALP). Lecture Notes in Computer Science, vol. 3142, pp. 196–207. Springer, Berlin (2004)
4. Englert, M., Westermann, M.: Lower and upper bounds on FIFO buffer management in qos switches. In: Azar, Y., Erlebach, T. (eds.) Algorithms – ESA 2006, 14th Annual European Symposium, Proceedings. Lecture Notes in Computer Science, vol. 4168, pp. 352–363. Springer, Berlin (2006)
5. Jawor, W.: Three dozen papers on online algorithms. SIGACT News **36**(1), 71–85 (2005)
6. Kesselman, A., Lotker, Z., Mansour, Y., Patt-Shamir, B., Schieber, B., Sviridenko, M.: Buffer overflow management in QoS switches. SIAM J. Comput. **33**(3), 563–583 (2004)
7. Kesselman, A., Mansour, Y., van Stee, R.: Improved competitive guarantees for QoS buffering. In: Di Battista, G., Zwick, U. (eds.) Algorithms – ESA 2003, Proceedings Eleventh Annual European Symposium. Lecture Notes in Computer Science, vol. 2380, pp. 361–373. Springer, Berlin (2003)
8. Lotker, Z., Patt-Shamir, B.: Nearly optimal FIFO buffer management for DiffServ. In: Proceedings of the 21st ACM Symposium on Principles of Distributed Computing (PODC 2002), pp. 134–142. ACM, New York (2002)
9. Mansour, Y., Patt-Shamir, B., Lapid, O.: Optimal smoothing schedules for real-time streams. In: Proc. 19th Symp. on Principles of Distributed Computing (PODC), pp. 21–29. ACM, New York (2000)

PAC Learning

1984; Valiant

JOEL RATSABY
Department of Electrical and Electronic Engineering,
Ariel University Center of Samaria, Ariel, Israel

Keywords and Synonyms

Probably approximately correct learning

Problem Definition

Valiant's work defines a model for representing the general problem of learning a Boolean concept from examples. The motivation comes from classical fields of artificial intelligence [2], pattern classification [5] and machine learning [10]. Classically, these fields have employed numerous heuristics for representing knowledge and defining criteria by which computer algorithms can learn. The pioneering work of [12,13] provided the leap from heuristic-based approaches to a rigorous statistical theory of pattern recognition (see also [1,4,11]). Their main contribution was the introduction of probabilistic upper bounds on the generalization error which hold uniformly over a whole class of concepts. Valiant's main contribution is in formalizing this probabilistic theory into a general model for computational inference. This model which is known as the *Probably Approximately Correct* (PAC) model of learnability is concerned with computational complexity of learning. In his formulation, learning is depicted as an interaction between a teacher and a learner with two main procedures, one which provides randomly drawn examples x of the concept c that is being learned and the second acts as an oracle which provides the correct classification label $c(x)$. Based on a finite number of such examples drawn identically and independently according to *any* fixed probability distribution, the aim of the learner is to infer an approximation of c which is correct with high confidence. Using the terminology of [9] suppose X denotes the space of instances, i. e., objects which a learner can obtain as training examples. A *concept* over X is a Boolean mapping from X to $\{0, 1\}$. Let $\mathbb{P}$ be any fixed probability distribution over X and c a fixed *target* concept to be learned. For any hypothesis concept h over X define by $L(h) = \mathbb{P}(c(x) \neq h(x))$ the *error* of h, i. e., the probability that h disagrees with c on a test instance x which is drawn according to $\mathbb{P}$. Then according to Valiant, an algorithm $\mathbb{A}$ for learning c is one which runs in time t and with a sample of size m where both t and m are polynomials with respect to some parameters (to be specified below) and produces a hypothesis concept h such that with high confidence $L(h)$ is small.

Key Results

The main result of Valiant's work is a formal definition of what constitutes a *learnable* problem. Formally, this is stated as follows: Let $\mathcal{H}$ be a class of concepts over X. Then $\mathcal{H}$ is *learnable* if there exists an algorithm $\mathbb{A}$ with the following property: for every possible target concept $c \in \mathcal{H}$, for every probability distribution $\mathbb{P}$ on X (this is sometimes referred to as the 'distribution-independence' assumption), for all values of a confidence parameter $0 < \delta < 1/2$ and an approximation accuracy parameter $0 < \epsilon < 1/2$, if $\mathbb{A}$ receives as input the value of δ, ϵ and a sample $S = \{(x_i, c(x_i))\}_{i=1}^{m}$ of cardinality m (which may depend on ε and δ) which consists of examples x_i that are randomly drawn according to $\mathbb{P}$ and labeled by an oracle as $c(x_i)$ then with probability $1 - \delta$, $\mathbb{A}$ outputs a hypothesis concept $h \in \mathcal{H}$ such that the error $L(h) \leq \epsilon$. That ϵ can be arbitrarily close to zero follows from what is known as the 'noise-free' assumption, i. e., that the labels comprise the true value of the target concept. If $\mathbb{A}$ runs in time t and if t and m are polynomial in $1/\epsilon$ and $1/\delta$ then $\mathcal{H}$ is *efficiently* PAC learnable.

Valiant has shown that the following classes are all PAC learnable: class of conjunctive normal form expressions with a bounded number of literals in each clause, the class of monotone disjunctive normal form expressions (here the learner requires in addition to S also an oracle that can answer membership queries, i. e., provide the true label $c(x)$ for an x in question), and the class of arbitrary expressions in which each variable occurs just once (using more powerful oracles). Work following Valiant's paper (see [8] for references) has shown that the classes of k-DNF, k-CNF and k-decision lists are PAC learnable for each fixed k. The class of concepts in the form of a disjunction of two conjunctions is not PAC learnable and neither is the class of existential conjunctive concepts on structural instance spaces with two objects. Linear threshold concepts (perceptrons) are PAC learnable on both Boolean and real-valued instance spaces but the class of concepts in the form of a conjunction of two linear threshold concepts is not PAC learnable. The same holds for disjunctions and linear thresholds of linear thresholds (i. e., multilayer perceptrons with two hidden units). If the weights are restricted to 1 and 0 (but the threshold is arbitrary) then linear threshold concepts on Boolean instances spaces are not PAC learnable.

It should be noted that the notion of PAC learnability discussed throughout this entry is sometimes referred to as "proper" PAC learnability because of the requirement that, when learning a concept class $\mathcal{H}$, the learning algorithm must output a hypothesis that also belongs to $\mathcal{H}$. Several of the negative results mentioned above can be circumvented in a model of "improper" PAC learning, where the learning algorithm is allowed to output hypotheses from a broader class of functions than $\mathcal{H}$. See [9] and the proceedings of the COLT conferences for many results of this type.

Applications

Valiant's paper is a milestone in the history of the area known as *Computational Learning Theory* (see proceedings of COLT conferences). The PAC model has been criticized in that the distribution independence assumption and the notion of target concepts with noise free training data are unrealistic in practice, e. g., in machine learning and AI. There has thus been much work on learning models that relax several of the assumptions in Valiant's PAC model. For instance, models which allow noisy labels or remove the assumptions on the independence of training examples, relax the assumption on the probability distribution to be fixed, allow the bounds to be distribution dependent, permit the training sample to be picked by the learner and labeled by the oracle instead of the random sample, or chosen by a helpful teacher. For references, see Sect. 2.6 of [1]. An important followup of Valiant's model was the work of [3] who unified his model with the uniform convergence results of [13]. They showed the important dependence between the notion of learnability and certain combinatorial properties of concept classes, one of which is known as the Vapnik-Chervonenkis (VC) dimension (see Sect. 3.4 of [1] for history on the VC-dimension).

Cross References

- ► Attribute-Efficient Learning
- ► Hardness of Proper Learning
- ► Learning with the Aid of an Oracle
- ► Learning Constant-Depth Circuits
- ► Learning DNF Formulas
- ► Learning with Malicious Noise

Recommended Readings

For a recommended collection of works on the PAC model and its extensions see [6,7,8].

1. Anthony, M., Bartlett, P.L.: Neural Network Learning: Theoretical Foundations. Cambridge University Press, Cambridge, England (1999)
2. Barr, A., Feigenbaum, E.A.: The Handbook of Artificial Intelligence. Addison-Wesley Pub (Sd) (1994)
3. Blumer, A., Ehrenfeucht, A., Haussler, D., Warmuth, M.: Learnability and the Vapnik–Chervonenkis dimension. J. ACM **36**(4), 929–965 (1989)
4. Devroye, L., Gyorfi, L., Lugosi, G.: A Probabilistic Theory of Pattern Recognition. Springer, New York, USA (1996)
5. Duda, R.O., Hart, P.E., Stork, D.G.: Pattern Classification. Wiley-Interscience Publication (2000)
6. Haussler, D.: Applying valiants learning framework to ai concept learning problems. In: Michalski, R., Kodratoff, Y. (eds.) Machine Learning: An Artificial Intelligence Approach. Morgan Kaufmann
7. Haussler, D.: Decision theoretic generalizations of the PAC model for neural net and other learning applications. Inf. Comput. **100**(1), 78–150 (1992)
8. Haussler, D.: Probably approximately correct learning and decision-theoretic generalizations. In: Smolensky, P., Mozer, M., Rumelhart, D. (eds.) Mathematical Perspectives on Neural Networks, pp. 651–718. L. Erlbaum Associates, Mahwah, New Jersey (1996)
9. Kearns, M.J., Vazirani, U.V.: An Introduction to Computational Learning Theory. M.I.T. Press, London, England (1997)
10. Mitchell, T.: Machine Learning. McGraw Hill (1997)
11. Pearl, J.: Capacity and error-estimates for boolean classifiers with limited complexity. IEEE Trans. on Pattern Recognition and Machine Intelligence, PAMI-**1**(4), 350–356 (1979)
12. Vapnik, V.N.: Estimations of dependences based on statistical data. Springer (1982)
13. Vapnik, V.N., Chervonenkis, A.Y.: On the uniform convergence of relative frequencies of events to their probabilities. Theory Probab. Apl. **16**, 264–280 (1971)

PageRank Algorithm

1998; Brin, Page

MONIKA HENZINGER
Google Switzerland & Ecole Polytechnique Federale de Lausanne (EPFL), Lausanne, Switzerland

Problem Definition

Given a user query current web search services retrieve all web pages that contain the query terms, resulting in a huge number of web pages for the majority of searches. Thus it is crucial to reorder or *rank* the resulting documents with the goal of placing the most relevant documents first. Frequently, ranking uses two types of information: (1) query-specific information and (2) query-independent information. The query-specific part tries to measure how relevant the document is to the query. Since it depends to a large part on the content of the page, it is mostly under the control of the page's author. The query-independent information tries to estimate the quality of the page in general. To achieve an objective measure of page quality it is important that the query-independent information incorporates a measure that is not controlled by the author. Thus the problem is to find a measure of page quality that (a) cannot be easily manipulated by the web page's author and (b) works well for *all* web pages. This is challenging as web pages are extremely heterogeneous.

Key Results

The hyperlink structure of the web is a good source for basing such a measure as it is hard for one author or a small set of authors to influence the whole structure, even though they can manipulate a subset of the web pages. Brin and Page showed that a relatively simple analysis of the hyperlink structure of the web can be used to produce a quality measure for web documents that leads to large improvements in search quality. The measure is called the *PageRank* measure.

Linear Algebra-based Definition

Let n be the total number of web pages. The PageRank vector is an n-dimensional vector with one dimension for each web page. Let d be a small constant, like 1/8, let $deg(p)$ denote the number of hyperlinks in the body text of page p and let $PR(p)$ denote the PageRank value of page p. Assume first that every page contains at least one hyperlink. In such a collection of web pages the PageRank vector is computed by solving a system of linear equations that contains for each page p the equation

$$PR(p) = d/n + (1-d) * \sum_{q \text{ has hyperlink to } p} PR(q)/deg(q)\,.$$

In matrix notation the PageRank vector is the Eigenvector with 1-Norm one of the matrix A with $d/n+(1-d)/deg(q)$ for entryA_{qp} if q has a hyperlink to p and d/n otherwise.

If web pages without hyperlinks are allowed in this linear system then they might become "PageRank sinks", i. e., they would "receive" PageRank from the pages pointing to them, but would not "give out" their PageRank, potentially resulting in an "unusually high" PageRank value for themselves. Brin and Page proposed two ways to deal with web pages without out-links, namely either to recursively delete them until no such web pages exist anymore in the collection or to add a hyperlink from each such page to *every* other page.

Random Surfer Model

Let the *web graph* $G = (V, E)$ be a directed graph such that each node corresponds to a web page and every hyperlink corresponds to a directed edge from the referencing node to the referenced node. The PageRank can also be interpreted as the following random walk in the web graph. The random walk starts at a random node in the graph. Assume in step k it visits page q. Then it flips a biased coin and with probability d or if q has no out-edges, it selects a random node out of V and visits it in step $k+1$. Otherwise it selects a random out-edge of the current node and visits it in step $k+1$. (Note that this corresponds to adding a directed edge from every page without hyperlink to *every* node in the graph.) Under certain conditions (which do not necessarily hold on the web) the stationary distribution of this random walk corresponds to the PageRank vector. See [1,4] for details.

Brin and Page also suggested to compute the PageRank vector approximately using the power method, i. e., by setting all initial values to $1/n$ and then repeatedly using the PageRank vector of the previous iteration to compute the PageRank vector of the current iteration using the above linear equations. After a hundred iterations barely any values change and the computation is stopped.

Applications

The PageRank measure is used as one of the factors by Google in its ranking of search results. The PageRank computation can be applied to other domains as well. Two examples are reputation management in P2P networks and learning word dependencies in natural language processing. In relational databases PageRank was used to weigh database tuples in order to improve keyword searching when a user does not know the schema. Finally, in rank aggregation PageRank can be used to find a permutation that minimally violates a set of given orderings. See [1] for more details.

Variations of PageRank were studied as well. Personalizing the PageRank computation such that the values reflect the interest of a user has received a lot of attention. See [3] for a survey on this topic. It can also be modified to be used for detecting web search spam, i. e., web pages that try to manipulate web search results [1].

Recommended Reading

1. Berkhin, P.: A survey on PageRank computing. Internet Math. **2**(1), 73–120 (2005)
2. Brin, S., Page, L.: The anatomy of a large-scale hypertextual Web search engine. In: Proc. 7th Int. World Wide Web Conference, pp. 107–117. Elsevier Science, Amsterdam (1998)
3. Haveliwala, T., Kamvar, S., Jeh, G.: An Analytical Comparison of Approaches to Personalizing PageRank. In: Technical Report. Stanford University, Stanford (2003)
4. Langville, A.N., Meyer, C.D.: Deeper Inside PageRank. Internet Math. **1**(3), 335–380 (2004)
5. Page, L., Brin, S., Motwani, R., Winograd, T.: The PageRank Citation Ranking: Bringing Order to the Web. In: Technical Report. Stanford University, Stanford (1998)

Paging

1985; Sleator, Tarjan, Fiat, Karp, Luby, McGeoch, Sleator, Young
1991; Sleator, Tarjan, Fiat, Karp, Luby, McGeoch, Sleator, Young

Rob van Stee
Department of Computer Science,
University of Karlsruhe, Karlsruhe, Germany

Keywords and Synonyms

Caching

Problem Definition

Computers generally have a small amount of fast memory to keep important data readily available. This is known as the *cache*. The question which is considered in this chapter is which pages should be kept in the cache when a new page is requested.

Formally, a two-level store of memory is considered. The cache can contain k pages, and the slow memory can contain n pages, where typically n is much larger than k. The input is a sequence of requests to pages. Whenever a requested page is not in the cache, the algorithm incurs a fault. The goal is to minimize the total number of page faults.

It is easy to give an optimal algorithm if the whole request sequence is known: on each fault, evict that page from the cache which is next requested the furthest in the future [2]. However, in practice, paging decisions need to be made without knowledge of the future. Thus an *online* algorithm is needed, which makes its decisions for each request based only on that request and previous requests.

Key Results

A major contribution of the paper of Sleator and Tarjan [6] was the idea of *competitive analysis*. In this type of analysis, the performance of an online algorithm is compared to that of an optimal offline algorithm OPT. Thus the offline algorithm knows the entire input and moreover it can use unbounded computational resources to find the best possible solution for this input.

Denote the cost of an algorithm ALG on an input sequence σ by $\text{ALG}(\sigma)$. An online algorithm $\mathcal{A}$ is called c-competitive if there exists a constant b such that on every request sequence σ,

$$\mathcal{A}(\sigma) \leq c \cdot \text{OPT}(\sigma) + b \,. \tag{1}$$

The competitive ratio of $\mathcal{A}$ is the smallest value of c such that $\mathcal{A}$ is c-competitive. This definition is very similar to that of the approximation ratio of approximation algorithms. However, it should be noted that there are no computational restrictions on the online algorithm. In particular, it is allowed to use exponential time to make its decisions. Thus the competitive ratio purely measures the performance loss that results from not knowing the future.

Using this definition, Sleator and Tarjan give tight bounds on the best competitive ratio which can be achieved by a deterministic algorithm. They show that two well-known algorithms both have a competitive ratio of k:

- FIFO (First In First Out), which on a fault evicts the page that was loaded into the cache the earliest
- LRU (Least Recently Used), which on a fault evicts the page that was *requested* least recently.

Additionally, they show that any deterministic algorithm has a competitive ratio of at least k, implying that k is the best value that can be achieved.

Fiat et al. [3] considered *randomized* paging algorithms. A randomized online algorithm is allowed to use random bits in its decision making. To measure its performance, consider the expectation of the cost for a particular input sequence and compare that to the optimal cost for that sequence. Thus a randomized online algorithm is c-competitive if there exists a constant b such that on every request sequence σ,

$$\mathbb{E}(\mathcal{A}(\sigma)) \leq c \cdot \text{OPT}(\sigma) + b \,. \tag{2}$$

Fiat et al. presented the marking algorithm. This algorithm marks pages that are requested. On a fault, an unmarked page is selected uniformly at random and evicted from the cache. When all pages are marked and a fault occurs, it unmarks all pages and then evicts one uniformly at random.

Fiat et al. showed that this algorithm is $2H_k$-competitive. Here H_k is the kth harmonic number: $H_k = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{k}$. It is known that $\ln(k+1) \leq H_k \leq \ln(k) + 1$. They also showed that no randomized paging algorithm can have a competitive ratio less than H_k. Thus the marking algorithm is at most twice as bad as the best possible online algorithm (with regard to the competitive ratio). A randomized algorithm with competitive ratio exactly H_k was given by McGeoch and Sleator [4]. This algorithm is much more complicated than the marking algorithm.

Applications

Memory management has long been and continues to be a fundamentally important problem in computing systems. In particular, the question of how to manage a two-level or multilevel store of memory remains crucial to the performance of computers, from the simplest personal or game computer to the largest servers.

The study of the paging problem also was very important for the development of the whole area of online algorithms. The paper by Sleator and Tarjan formally introduced the concept of the competitive ratio as a performance measure for online algorithms. This ratio is in wide use today.

Open Problems

The problem as presented in this chapter is closed, since an upper and a lower bound of k for deterministic algorithms and an upper and a lower bound of H_k for randomized algorithms are obtained.

Variations of this problem continue to inspire new research. The basic problem has also been further studied, because the upper bound of k for LRU is disappointingly high and it is known from practice that LRU is "really" constant competitive. Recently, Panagiotou and Souza [5] managed to give a theoretical justification for this observation by formally restricting the input sequences to be closer to the ones that occur in practice. Additional justification for using LRU was given by Angelopoulos et al. [1], using a direct comparison of LRU to all other online algorithms.

Cross References

► Analyzing Cache Misses
► Deterministic Searching on the Line
► Online Paging and Caching

Recommended Reading

1. Angelopoulos, S., Dorrigiv, R., López-Ortiz, A.: On the separation and equivalence of paging strategies. In: Proceedings of the 18th Annual ACM–SIAM Symposium on Discrete Algorithms. ACM/SIAM, New York, Philadelphia (2007)
2. Belady, L.A.: A study of replacement algorithms for virtual storage computers. IBM Syst. J. **5**, 78–101 (1966)
3. Fiat, A., Karp, R., Luby, M., McGeoch, L.A., Sleator, D., Young, N.E.: Competitive paging algorithms. J. Algorithms **12**, 685–699 (1991)
4. McGeoch, L., Sleator, D.: A strongly competitive randomized paging algorithm. Algorithmica **6**(6), 816–825 (1991)
5. Panagiotou, K., Souza, A.: On adequate performance measures for paging. In: STOC '06: Proceedings of the thirty-eighth annual ACM symposium on Theory of computing, pp. 487–496. ACM Press, New York, NY, USA (2006)
6. Sleator, D., Tarjan, R.E.: Amortized efficiency of list update and paging rules. Commun. ACM **28**, 202–208 (1985)

Parallel Algorithms for Two Processors Precedence Constraint Scheduling

2003; Jung, Serna, Spirakis

Maria Serna
Department of Language & System Information, Technical University of Catalonia, Barcelona, Spain

Keywords and Synonyms

Optimal scheduling for two processors

Problem Definition

In the general form of *multiprocessor precedence scheduling problems* a set of n tasks to be executed on m processors is given. Each task requires exactly one unit of execution time and can run on any processor. A directed acyclic graph specifies the precedence constraints where an edge from task x to task y means task x must be completed before task y begins. A solution to the problem is a schedule of shortest length indicating when each task is started. The work of Jung, Serna, and Spirakis provides a parallel algorithm (on a PRAM machine) that solves the above problem for the particular case that $m = 2$, that is where there are two parallel processors.

The *two processor precedence constraint scheduling problem* is defined by a directed acyclic graph (dag) $G = (V, E)$. The vertices of the graph represent unit time tasks, and the edges specify precedence constraints among the tasks. If there is an edge from node x to node y then x is an *immediate predecessor* of y. *Predecessor* is the transitive closure of the relation immediate predecessor, and *successor* is its symmetric counterpart. A *two processor schedule* is an assignment of the tasks to time units $1, \dots, t$ so that each task is assigned exactly one time unit, at most two tasks are assigned to the same time unit, and if x is a predecessor of y then x is assigned to a lower time unit than y. The length of the schedule is t. A schedule having minimum length is an *optimal* schedule. Thus the problem is the following:

Name	Two processor precedence constraint scheduling
Input	A directed acyclic graph
Output	A minimum length schedule preserving the precedence constraints.

Preliminaries

The algorithm assume that tasks are partitioned into levels as follows:

(i) Every task will be assigned to only one level
(ii) Tasks having no successors will be assigned to level 1 and
(iii) For each level i, all tasks which are immediate predecessors of tasks in level i will be assigned to level $i + 1$.

Clearly topological sort will accomplish the above partition, and this can be done by an NC algorithm that uses $O(n^3)$ processors and $O(\log n)$ time, see [3]. Thus, from now on, it is assumed that a level partition is given as part of the input. For sake of convenience two special tasks, t_0 and t^* are added, in such a way that the original graph could be taught as the graph formed by all tasks that are successors of t_0 and predecessors of t^*. Thus t_0 is a predecessor of all tasks in the system (actually an immediate predecessor of tasks in level the highest level $L(G)$) and t^* is a successor of all tasks in the system (an immediate successor of level 1 tasks).

Notice that if two tasks are at the same level they can be paired. But when x and y are at different levels, they can be paired only when neither of them is a predecessor of the other. Let $L(G)$ denote the number of levels in a given precedence graph G. A *level schedule* schedules tasks level by level. More precisely, suppose levels $L(G), \ldots, i+1$ have already been scheduled and there are k unscheduled tasks remaining on level i. When k is even, those tasks with are paired with each other. When k is odd, $k-1$ of the tasks are paired with each other, while the remaining task may (but not necessarily) be paired with a task from a lower level.

Given a level schedule level *i jumps to level i'* ($i' < i$) if the last time step containing a task from level i also contains a task from level i'. If the last task from level i is scheduled with an empty slot, it is said that level i *jumps to level* 0. The *jump sequence* of a level schedule is the list of levels jumped to. A *lexicographically first jump schedule* is a level schedule whose jump sequence is lexicographically greater than any other jump sequence resulting from a level schedule.

Given a graph G a *level partition* of G is a partition of the nodes in G into two sets in such a way that levels $0, \ldots, k$ are contained in one set (the upper part) denoted by U, and levels $k+1, \ldots, L$ in the other (the lower part) denoted by L. Given a graph G and a level i, the *i-partition* of G (or the partition at level i) is formed by the graphs U_i and L_i defined as U_i contains all nodes x such that level$(x) < i$ and L_i contains all nodes x with level$(x) > i$. Note that each i-partition determines two different level partitions depending on whether level i nodes are assigned to the upper or the lower part. A task $x \in U_i$ is called *free* with respect to a partition at level i if x has no predecessors in L_i.

Auxiliary Problems

The algorithm for the two processors precedence constraint scheduling problem uses as a building block an algorithm for solving a matching problem in a particular graph class.

A *full convex bipartite graph* G is a triple (V, W, E), where $V = \{v_1, \ldots, v_k\}$ and $W = \{w_1, \ldots, w_{k'}\}$ are disjoint sets of vertices. Furthermore the edge set E satisfies the following property: If $(v_i, w_j) \in E$ then $(v_q, w_j) \in E$ for all $q \geq i$. Thus, from now on it is assumed that the graph is connected.

A set $F \subseteq E$ is a *matching* in the graph $G = (V, W, E)$ iff no two edges in F have a common endpoint. A *maximal matching* is a matching that cannot be extended by the addition of any edge in G. A *lexicographically first maximal matching* is a maximal matching whose sorted list of edges is lexicographically first among all maximal matchings in G.

Key Results

When the number of processors m is arbitrary the problem is known to be NP-complete [8]. For any $m \geq 3$, the complexity is open [6]. Here the case of interest has been $m = 2$. For two processors a number of efficient algorithms has been given. For sequential algorithms see [2,4,5] among others. The first deterministic parallel algorithm was given by Helmbold and Mayr [7], thus establishing membership in the class NC. Previously [9] gave a randomized NC algorithm for the problem. Jung, Serna and Spirakis present a new parallel algorithm for the two processors scheduling problem that takes time $O(\log^2 n)$ and uses $O(n^3)$ processors on a CREW PRAM. The algorithm improves the number of processors of the algorithm given in [7] from $O(n^7 L(G)^2)$, where $L(G)$ is the number of levels in the precedence graph, to $O(n^3)$. Both algorithms compute a level schedule that has a lexicographically first jump sequence.

To match jumps with tasks it is used a solution to the problem of computing the lexicographically first matching for a special type of convex bipartite graphs, here called *full convex bipartite graphs*. A geometric interpretation of this problem leads to the discovery of an efficient parallel algorithm to solve it.

Theorem 1 *The lexicographically first maximal matching of full convex bipartite graphs can be computed in time $O(\log n)$ on a CREW PRAM with $O(n^3/\log n)$ processors, where n is the number of nodes.*

The previous algorithm is used to solve efficiently in parallel two related problems.

Theorem 2 *Given a precedence graph G, there is a PRAM parallel algorithm that computes all levels that jump to level 0 in the graph L_i and all tasks in level $i-1$ that can be scheduled together with a task in level i, for $i = 1, \ldots, L(G)$, using $O(n^3)$ processors and $O(\log^2 n)$ time.*

Theorem 3 *Given a level partition of a graph G together with the levels in the lower part in which one task remains to be matched with some other task in the upper part of the graph. There is a PRAM parallel algorithm that computes the corresponding tasks in time $O(\log n)$ using $n^3/\log n$ processors.*

With those building blocks the algorithm for two processor precedence constraint scheduling starts by doing some preprocessing and after that an adequate decomposition that insure that at each recursive call a number of problems of half size are solved in parallel. This recursive schema is the following:

Algorithm Schedule

```
0. Preprocessing
1. Find a level $i$ such that $|U_i| \leq n/2$ and $|L_i| \leq n/2$.
2. Match levels that jump to free tasks in level $i$.
3. Match levels that jump to free tasks in $U_i$.
4. If level $i$ (or $i+1$) remain unmatched try to match it
   with a non free task.
5. Delete all tasks used to match jumps.
6. Apply (1)–(5) in parallel to $L_i$ and the modified $U_i$.
```

Algorithm **Schedule** stops whenever the corresponding graph has only one level.

The correction an complexity bounds for algorithm **Schedule** follows from the previous results, leading to:

Theorem 4 *There is an NC algorithm which finds an optimal two processors schedule for any precedence graph in time $O(\log^2 n)$ using $O(n^3)$ processors.*

Applications

A fundamental problem in many applications is to devise a proper schedule to satisfy a set of constrains. Assigning people to jobs, meetings to rooms, or courses to final exam periods are all different examples of scheduling problems. A key and critical algorithm in parallel processing is the one mapping tasks to processors. In the performance of such an algorithm relies many properties of the system, like load balancing, total execution time, etc. Scheduling problems differ widely in the nature of the constraints that must be satisfied, the type of processors, and the type of schedule desired.

The focus on precedence-constrained scheduling problems for directed acyclic graphs has a most direct practical application in problems arising in parallel processing. In particular in systems where computations are decomposed, prior to scheduling into approximately equal sized tasks and the corresponding partial ordering among them is computed. These constraints must define a directed acyclic graph, acyclic because a cycle in the precedence constraints represents a Catch situation that can never be resolved.

Open Problems

The parallel deterministic algorithm for the two processors scheduling problem presented here improves the number of processors of the Helmbold and Mayr algorithm for the problem [7]. However, the complexity bounds are far from optimal: recall that the sequential algorithm given in [5] uses time $O(e + n\alpha(n))$, where e is the number of edges in the precedence graph and $\alpha(n)$ is an inverse Ackermann's function. Such an optimal algorithm might have a quite different approach, in which the levelling algorithm is not used.

Interestingly enough computing the lexicographically first matching for full convex bipartite graphs is in NC, in contraposition with the results given in [1] which show that many problems defined through a lexicographically first procedure in the plane are P-complete. It is an interesting problem to show whether all these problems fall in NC when they are convex.

Cross References

- ▶ List Scheduling
- ▶ Maximum Matching
- ▶ Minimum Makespan on Unrelated Machines
- ▶ Shortest Elapsed Time First Scheduling
- ▶ Stochastic Scheduling
- ▶ Voltage Scheduling

Recommended Reading

1. Attallah, M., Callahan, P., Goodrich, M.: P-complete geometric problems. Int. J. Comput. Geom. Appl. **3**(4), 443–462 (1993)
2. Coffman, E.G., Graham, R.L.: Optimal scheduling for two processors systems. Acta Informatica **1**, 200–213 (1972)
3. Dekel, E., Nassimi, D., Sahni, S.: Parallel matrix and graph algorithms. SIAM J. Comput. **10**, 657–675 (1981)
4. Fujii, M., Kasami, T., Ninomiya, K.: Optimal sequencing of two equivalent processors. SIAM J. Comput. **17**, 784–789 (1969)
5. Gabow, H.N.: An almost linear time algorithm for two processors scheduling. J. ACM **29**(3), 766–780 (1982)
6. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the theory of NP completeness. Freeman, San Francisco (1979)
7. Helmbold, D., Mayr, E.: Two processor scheduling is in NC. SIAM J. Comput. **16**(4), 747–756 (1987)
8. Ullman, J.D.: NP-complete scheduling problems. J. Comput. Syst. Sci. **10**, 384–393 (1975)
9. Vazirani, U., Vazirani, V.: Two-processor scheduling problem is in random NC. SIAM J. Comput. **18**(4), 1140–1148 (1989)

Parallel Connectivity and Minimum Spanning Trees

2001; Chong, Han, Lam

TAK-WAH LAM
Department of Computer Science,
University of Hong Kong, Hong Kong, China

Keywords and Synonyms

EREW PRAM algorithms for finding connected components and minimum spanning trees

Problem Definition

Given a weighted undirected graph G with n vertices and m edges, compute a minimum spanning tree (or spanning forest) of G on a parallel random access machine (PRAM) without concurrent write capability.

A minimum spanning tree of a graph is a spanning tree with the smallest possible sum of edge weights. Parallel random access machine (PRAM) is an abstract model for designing parallel algorithms and understanding the power of parallelism. In this model, processors (each being a random access machine) work in a synchronous manner and communicate through a shared memory. PARM can be further classified according to whether it is allowed for more than one processor to read and write into the same shared memory location simultaneously. The strongest model is CRCW (concurrent-read, concurrent-write) PRAM, and the weakest is EREW (exclusive-read, exclusive-write) PRAM. For an introduction to PRAM algorithms, one can refer to Karp and Ramachandran [8] and JáJá [5].

The input graph G is assumed to be given in the form of adjacency lists. Furthermore, isolated (degree-0) vertices are removed and hence it is assumed that $m \geq n$.

Key Results

The MST problem is related to the connected component (CC) problem, which is to find the connected components of an undirected graph. Sequential algorithms for solving the CC problem and the MST problem in $O(m)$ time and $O(m \log n)$ time, respectively, were known a few decades ago. A number of more efficient MST algorithms have since been published, the most recent of which is Pettie and Ramachandran's algorithm [9], which is provably optimal.

In the parallel context, both problems are often solved in a similar way. With respect to CRCW PRAM, the two problems can be solved using $O(\log n)$ time and $n + m$ processors (see, e. g., Cole and Vishkin [3]). Using randomization, $(n + m)/\log n$ processors are sufficient to solve these problems in $O(\log n)$ expected time [2,10].

For EREW PRAM, $O(\log^2 n)$ time algorithms for the CC and MST problems were developed in the early 80's. For a while, it was believed that the exclusive write models (including both concurrent read and exclusive read) could not overcome the $O(\log^2 n)$ time bound [8]. The first breakthrough was due to Johnson and Metaxas [6]; they devised $O(\log^{1.5} n)$ time algorithms for the CC problem and the MST problem. These results were soon improved to $O(\log n \log\log n)$ time by Chong and Lam. If randomization is allowed, the time complexity can be improved to $O(\log n)$ expected time and optimal work [7,10,11]. Finally, Chong, Han, and Lam [1] obtained an algorithm for MST (and CC) using $O(\log n)$ time and $n + m$ processors. This algorithm does not need randomization. Notice that $\Theta(\log n)$ is optimal since these graphs problems are at least as hard as computing the OR of n bits, and Cook et al. [4] have proven that the latter requires $\Omega(\log n)$ time on exclusive-write PRAM no matter how many processors are used.

Below is a sketch of some ideas for computing a minimum spanning tree in parallel without using concurrent write.

Without loss of generality, assume that the edge weights are all distinct. Thus, G has a unique minimum spanning tree, which is denoted by T_G^*. Let B be a subset of edges in G which contains no cycle. B induces a set of trees $F = \{T_1, T_2, \cdots, T_l\}$ in a natural sense—two vertices in G are in the same tree if they are connected by an edge of B. B is said to be a λ-forest if each tree $T \in F$ has at least λ vertices. For example, if B is the empty set then B is a 1-forest; a spanning tree is an n-forest.

Suppose that B is a λ-forest and its edges are all found in T_G^*. Then B can be augmented to give a 2λ-forest using a greedy approach: Let F' be an arbitrary subset of F including all trees $T \in F$ with fewer than 2λ vertices. For every tree in F', pick its minimum external edge (i. e., the smallest-weight edge connecting to a vertex outside the tree). Denote B' as this set of edges. It can be proven that B' consists of edges in T_G^* only, and $B \cup B'$ is a 2λ-forest. The above idea allows us to find T_G^* in $\lfloor \log n \rfloor$ stages as follows.

```
1. $B \leftarrow \phi$
2. **For** $i = 1$ to $\lfloor \log n \rfloor$ **do** /* Stage $i$ */
   (a) Let $F$ be the set of trees induced by $B$ on $G$. Let $F'$
       be an arbitrary subset of $F$ such that $F'$ includes all
       trees $T \in F$ with fewer than $2^i$ vertices.
   (b) $B_i \leftarrow \{e \mid e$ is the minimum external edge of
       $T \in F'\}$; $B \leftarrow B \cup B_i$
3. **return** $B$
```

Different strategies for choosing the set F' in Step 1(a) may lead to different B_i's. Nevertheless, $B[1, i]$ is always a subset of T_G^* and induces a 2^i-forest. In particular, $B[1, \lfloor \log n \rfloor]$ induces exactly one tree, which is exactly T_G^*. Using standard parallel algorithmic techniques, each stage can be implemented in $O(\log n)$ time on EREW PRAM using a linear number of processors (see e. g. [5],). Therefore,

T^*_G can be found in $O(\log^2 n)$ time. In fact, most parallel algorithms for finding MST are based on a similar approach. These parallel algorithms are "sequential" in the sense that the computation of B_i starts only after B_{i-1} is available.

The $O(\log n)$-time EREW algorithm in [1], is based on some structural properties related to MST and can compute the B_i's in a more parallel fashion. In this algorithm, there are $\lfloor \log n \rfloor$ concurrent threads (a thread is simply a group of processors). For $1 \leq i \leq \lfloor \log n \rfloor$, Thread i aims at computing B_i, and it actually starts long before Thread $i-1$ has computed B_{i-1} and it receives the output of Threads 1 to $i-1$ (i. e., $B_1, \cdots, B_{i-1}$) incrementally. More specifically, the algorithm runs in $\lfloor \log n \rfloor$ supersteps, where each superstep lasts $O(1)$ time. Thread i delivers B_i at the end of the ith superstep. The computation of Thread i is divided into $\lfloor \log i \rfloor$ phases. Let us first consider a simple case when i is a power of two. Phase 1 of Thread i starts at the $(i/2+1)$th superstep, i. e., when $B_1, \cdots, B_{i/2}$ are available. Phase 1 takes no more than $i/4$ supersteps, ending at the $(i/2+i/4)$th superstep. Phase 2 starts at the $(i/2+i/4+1)$th superstep (i. e., when $B_{i/2+1}, \cdots, B_{i/2+i/4}$ are available) and uses $i/8$ supersteps. Each subsequent phase uses half as many supersteps as the preceding phase. The last phase (Phase $\log i$) starts and ends within the ith superstep; note that B_{i-} is available after $(i-1)$th superstep.

Applications

Finding connected components or MST is a key step in several parallel algorithms for other graph problems. For example, the Chong-Han-Lam algorithm implies an $O(\log n)$-time algorithm for finding ear decomposition and biconnectivity without using concurrent write.

Cross References

► Graph Connectivity
► Randomized Parallel Approximations to Max Flow

Recommended Reading

1. Chong, K.W., Han, Y., Lam, T.W.: Concurrent Threads and Optical Parallel Minimum Spanning Trees Algorithm. J. ACM **48**(2), 297–323 (2001)
2. Cole, R., Klein, P.N., Tarjan, R.E.: Finding minimum spanning forests in logarithmic time and linear work using random sampling. In: Proceedings of the 8th Annual ACM Symposium on Parallel Architectures and Algorithms, 1996, pp. 243–250
3. Cole, R., Vishkin, U.: Approximate and Exact Parallel Scheduling with Applications to List, Tree, and Graph Problems. In: Proceedings of the 27th Annual IEEE Symposium on Foundations of Computer Science, 1986, pp. 478–491
4. Cook, S.A., Dwork, C., Reischuk, R.: Upper and lower time bounds for parallel random access machines without simultaneous writes. SIAM J. Comput. **15**(1), 87–97 (1986)
5. JáJá, J.: An Introduction to Parallel Algorithms. Addison-Wesley (1992)
6. Johnson, D.B., Metaxas, P.: Connected Components in $O(\lg^{3/2} |V|)$ Parallel Time for the CREW PRAM. In: Proceedings of the 32nd Annual IEEE Symposium on Foundations of Computer Science, 1991, pp. 688–697
7. Karger, D.R.: Random sampling in Graph Optimization Problems. Ph. D. thesis, Department of Computer Science, Stanford University (1995)
8. Karp, R.M., Ramachandran, V.: Parallel Algorithms for Shared-Memory Machines. In: Van Leeuwen Ed, J. (ed) Handbook of Theoretical Computer Science, vol. A, pp. (869–941). MIT Press, Massachusetts (1990)
9. Pettie, S. Ramachandran, V.: An Optimal Minimum Spanning Tree Algorithm. J. ACM **49**(1), 16–34 (2002)
10. Pettie, S., Ramachandran, V.: A randomized time-work optimal parallel algorithm for finding a minimum spanning forest. SIAM J. Comput. **31**(6), 1879–1895 (2002)
11. Poon, C.K., Ramachandran, V.: A randomized linear-work EREW PRAM algorithm to find a minimum spanning forest. Algorithmica **35**(3), 257–268 (2003)

Parameterized Algorithms for Drawing Graphs

2004; Dujmovic, Whitesides

HENNING FERNAU
Institute for Computer Science, University of Trier, Trier, Germany

Problem Definition

ONE-SIDED CROSSING MINIMIZATION (OSCM) can be viewed as a specific form of drawing a bipartite graph $G = (V_1, V_2, E)$, where all vertices from partition V_i are assigned to the same line (also called layer) L_i in the plane, with L_1 and L_2 being parallel. The vertex assignment to L_1 is fixed, while that to L_2 is free and should be chosen in a way to minimize the number of crossings between etdes drawn as straight-line segments.

Notations

A graph G is described by its vertex set V and its edge set E, i. e., G=(V, E), with $E \subseteq V \times V$. The (open) *neighborhood* of a vertex v, denoted $N(v)$, collects all vertices that are adjacent to v. $N[v] = N(v) \cup \{v\}$ denotes the *closed neighborhood* of v. $\deg(v) = |N(v)|$ is the *degree* of v. For a vertex set S, $N(S) = \bigcup_{v \in S} N(v)$, and $N[S] = N(S) \cup S$. $G[S]$ denotes the graph induced by vertex set S, i. e., $G[S] = (S, E \cap (S \times S))$. A graph $G = (V, E)$ with vertex set

V and edge set $E \subseteq V \times V$ is *bipartite* if there is a partition of V into two sets V_1 and V_2 such that $V = V_1 \cup V_2$, $V_1 \cap V_2 = \emptyset$, and $E \subseteq V_1 \times V_2$. For clarity, $G = (V_1, V_2, E)$ is written in this case.

A *two-layer drawing* of a bipartite graph $G = (V_1, V_2, E)$ can be described by two linear orders $<_1$ on V_1 and $<_2$ on V_2. This drawing can be realized as follows: the vertices of V_1 are placed on a line L_1 (also called *layer*) in the order induced by $<_1$ and the vertices of V_2 are placed on a second layer L_2 (parallel to the first one) in the order induced by $<_2$; then, draw a straight-line segment for each edge $e = (u_1, u_2)$ in E connecting the points that represent u_1 and u_2, respectively. A *crossing* is a pair of edges $e = (u_1, u_2)$ and $f = (v_1, v_2)$ that cross in the realization of a two-layer drawing $(G, <_1, <_2)$. It is well-known that two edges cross if and only if $u_1 <_1 v_1$ and $v_2 <_2 u_2$; in other word, this notion is a purely combinatorical object, independent of the concrete realization of the two-layer drawing. $\mathrm{cr}(G, <_1, <_2)$ denotes the number of crossings in the described two-layer drawing. In the graph drawing context, it is of course desirable to draw graphs with few crossings. In its simplest (yet probably most important) form, the vertex order in one layer is fixed, and the aim is to minimize crossings by choosing an order of the second layer. Formally, this means:

Problem 1 (k–OSCM)
INPUT: *A simple n-vertex bipartite graph $G = (V_1, V_2, E)$ and a linear order $<_1$ on V_1, a nonnegative integer k (the parameter).*
OUTPUT: *If possible, a linear order $<_2$ on V_2 such that $\mathrm{cr}(G, <_1, <_2) \le k$. If no such order exists, the algorithm should tell so.*

Given an instance $G = (V_1, V_2, E)$ and $<_1$ of OSCM and two vertices $u, v \in V_2$,

$$c_{uv} = \mathrm{cr}(G[N[\{u,v\}]], <_1 \cap (N(\{u,v\}) \times N(\{u,v\})), \{(u,v)\}) \; .$$

Hence, the closed neighborhoods of u and v are considered when assuming the ordering $u <_2 v$.

Consider the following as a running example:

Example 1 In Fig. 1, a concrete drawing of a bipartite graph is shown. Is this drawing optimal with respect to the number of crossings, assuming the ordering of the upper layer being fixed? At some points, more than two edges cross; in that case, a number is shown to count the crossings. All crossings are emphasized by a surrounding box.

Let us now compute the *crossing number matrix* (c_{uv}) for this graph.

c_{uv}	a	b	c	d	e
a	–	4	5	0	1
b	1	–	1	0	0
c	3	3	–	0	1
d	3	2	3	–	1
e	2	3	2	0	–

The number of crossings in the given drawing can be hence computed as

$$c_{ab} + c_{ac} + c_{ad} + c_{ae} + c_{bc} + c_{bd} + c_{be} + c_{cd} + c_{ce} + c_{de} = 13 \; .$$

Key Results

Exact exponential-time algorithms are mostly interesting when dealing with problems for which no polynomial-time algorithm is expected to exist.

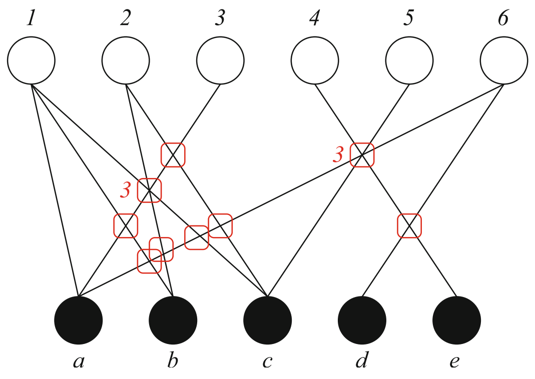
Theorem 1 ([6]) *The decision problem corresponding to k-OSCM is $\mathcal{NP}$-complete.*

In the following, to state the results, let $G = (V_1, V_2, E)$ be an instance of OSCM, where the ordering $<_1$ of V_1 is fixed.

It can be checked in polynomial time if an order of V_2 exists that avoids any crossings. This observation can be based on either of the following graph-theoretic characterizations:

Theorem 2 ([3]) $\mathrm{cr}(G, <_1, <_2) = 0$ *if and only if G is acyclic and, for every path (x, a, y) of G with $x, y \in V_1$, it holds: for all $u \in V_1$ with $x <_1 u <_1 y$, the only edge incident to u (if any) is (u, a).*

The previously introduced notion is crucial due to the following facts:



Parameterized Algorithms for Drawing Graphs, Figure 1
The running example for OSCM

Lemma 3 $\sum_{u,v\in V_2, u<_2 v} c_{uv} = \mathrm{cr}(G, <_1, <_2)$.

Theorem 4 ([9]) *If k is the minimum number of edge crossings in an OSCM instance $(G = (V_1, V_2, E), <_1)$, then*

$$\sum_{u,v\in V_2, u\neq v} \min\{c_{uv}, c_{vu}\} \leq k < 1.4664 \sum_{u,v\in V_2, u\neq v} \min\{c_{uv}, c_{vu}\} \,.$$

In fact, Nagamochi also presented an approximation algorithm with a factor smaller than 1.4664.

Furthermore, for any $u \in V_2$ with $\deg(u) > 0$, let l_u be the leftmost neighbor of u on L_1, and r_u be the rightmost neighbor of u. Two vertices $u, v \in V_2$ are called *unsuited* if there exists some $x \in N(u)$ with $l_v <_1 x <_1 r_v$, or there exists some $x \in N(v)$ with $l_u <_1 x <_1 r_u$. Otherwise, they are called *suited*. Observe that, for $\{u, v\}$ suited, $c_{uv} \cdot c_{vu} = 0$. Dujmović and Whitesides have shown:

Lemma 5 ([5]) *In any optimal ordering $<_2$ of the vertices of V_2, $u <_2 v$ is found if $r_u \leq_1 l_v$.*

This means that all suited pairs appear in their *natural ordering*.

This already allows us to formulate a first parameterized algorithm for OSCM, which is a simple search tree algorithm. In the course of this algorithm, a suitable ordering $<_2$ on V_2 is gradually constructed; when settling the ordering between u and v on V_2, $u <_2 v$ or $v <_2 u$ is *committed*. A *generalized instance* of OSCM therefore contains, besides the bipartite graph $G = (V_1, V_2, E)$, a partial ordering $<_2$ on V_2. A vertex $v \in V_2$ is *fully committed* if, for all $u \in V_2 \setminus \{u, v\}$, $\{u, v\}$ is committed.

Lemma 5 allows us to state the following rule:

RR1: For every pair of vertices $\{u, v\}$ from V_2 with $c_{uv} = 0$, commit $u <_2 v$. In the example, d would be fully committed by applying RR1, since the d-column in the crossing number matrix is all zeros; hence, ignore d in what follows.

Algorithm 1 is a simple search tree algorithm for OSCM that repeatedly uses Rule RR1.

Lemma 6 *OSCM can be solved in time $\mathcal{O}^*(2^k)$.*

Proof Before any branching can take place, the graph instance is reduced, so that every pair of vertices $\{u, v\}$ from V_2 which is not committed satisfies $\min\{c_{uv}, c_{vu}\} \geq 1$. Therefore, each recursive branch reduces the parameter by at least one. □

It is possible to improve on this very simple search tree algorithm. A first observation is that it is not necessary to branch at $\{x, y\} \subset V_2$ with $c_{xy} = c_{yx}$. This means two modifications to Algorithm 1:

- Line 5 should exclude $c_{xy} = c_{yx}$.
- Line 12 should arbitrary commit some $\{x, y\} \subset V_2$ that are not yet committed and recurse; only if all $\{x, y\} \subset V_2$ are committed, YES is to be returned.

These modifications immediately yield an $\mathcal{O}^*(1.6182^k)$ algorithm for OSCM. This is also the core of the algorithm proposed by Dujmović and Whitesides [5]. There, more details are discussed, as, for example:

- How to efficiently calculate all the crossing numbers c_{xy} in a preprocessing phase.
- How to integrate branch and cut elements in the algorithm that are surely helpful from a practical perspective.
- How to generalize the algorithm for instances that allow integer weights on the edges (multiple edges).

Further improvements are possible if one gives a deeper analysis of local patterns $\{x, y\} \in V_2$ such that $c_{xy}c_{yx} \leq 2$. This way, it has been shown:

Theorem 7 ([4]) *OSCM can be solved in time $\mathcal{O}^*(1.4656^k)$.*

A possible run of the improved search tree algorithm is displayed in Fig. 2, with the (optimal) outcome shown in Fig. 3.

Variants and Related Problems have been discussed in the literature.

1. Change the goal of the optimization: minimize the number of edges involved in crossings (ONE-LAYER PLANARIZATION (OLP)). As observed in [7,10], Theorem 2 almost immediately leads to an $\mathcal{O}^*(3^k)$ algorithm for OLP that was subsequently improved down to $\mathcal{O}^*(2^k)$ in [10].
2. One could allow more degrees of freedom by considering two (or more) layer assignments at the same time. For both the crossing minimization and the planarization variants, parameterized algorithms are reported in [3,7,10].
3. One can consider other additional constraints on the drawings or the admissible orderings; in [8], parameterized algorithms for two-layer assignment problems are discussed where the admissible orderings are restricted by binary trees.

Applications

Besides seeing the question of drawing bipartite graphs as an interesting problem in itself, e. g., for nice drawings of relational diagrams, this question quite naturally shows up in the so-called Sugiyama approach to hierarchical graph drawing, see [12]. This very popular approach tack-

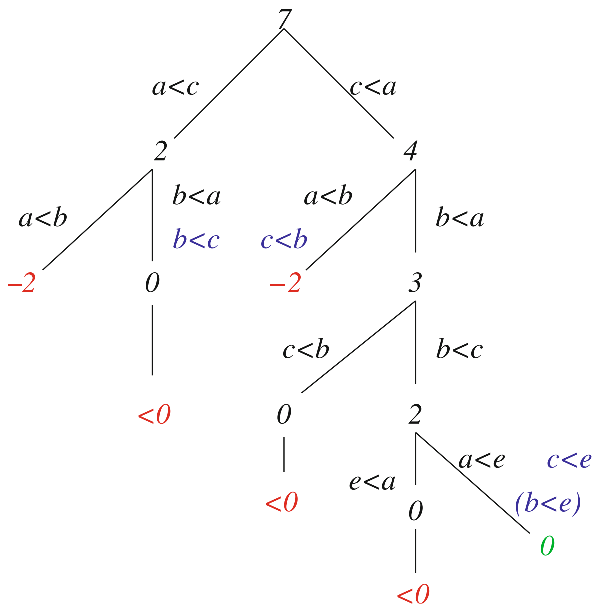
```
Require: a bipartite graph $G = (V_1, V_2, E)$, an integer $k$, a linear ordering $<_1$ on $V_1$, a partial ordering $<_2$ on $V_2$
Ensure: YES iff the given OSCM instance has a solution

    repeat
      Exhaustively apply the reduction rules, adjusting $<_2$ and $k$ accordingly.
      Determine the vertices whose order is settled by transitivity and adjust $<_2$ and $k$ accordingly.
    until there are no more changes to $<_2$ and to $k$
 5: if $k < 0$ or $<_2$ contains both $(x, y)$ and $(y, x)$ then
      return NO.
    else if $\exists\{x, y\} \subseteq V_2$ : neither $x <_2 y$ nor $y <_2 x$ is settled then
      if OSCM-ST-simple($G, k - 1, <_1, <_2 \cup \{(x, y)\}$) then
        return YES
10:   else
        return OSCM-ST-simple($G, k - 1, <_1, <_2 \cup \{(y, x)\}$)
      end if
    else
      return YES
15: end if
```

Parameterized Algorithms for Drawing Graphs, Algorithm 1
A search tree algorithm solving OSCM, called OSCM-ST-simple

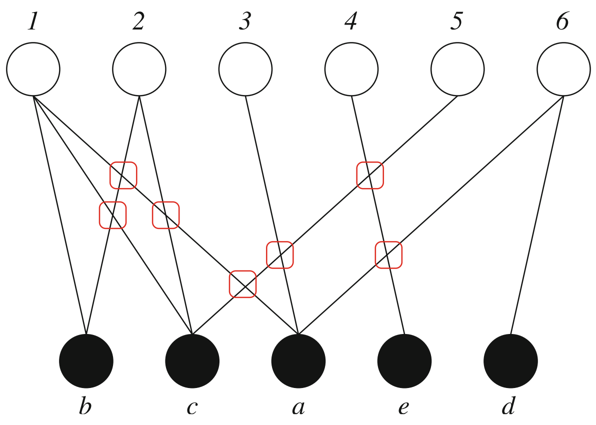
les the problem of laying out a hierarchical graph in three phases: (1) cycle removal (2) assignment of vertices to layers, (3) assignment of vertices to layers. The last phase is usually performed in a sweep-line fashion, intermediately solving many instances of OSCM. The third variant in the discussion above has important applications in computational biology.

Open Problems

As with all exponential-time algorithms, it is always a challenge to further improve on the running times of the algorithms or to prove lower bounds on those running times under reasonable complexity theoretic assumptions. Let us notice that the tacit assumptions underlying the approach by parameterized algorithmics are well met in this application scenario: e. g., one would not accept drawings with



Parameterized Algorithms for Drawing Graphs, Figure 2
A search tree example for OSCM



Parameterized Algorithms for Drawing Graphs, Figure 3
An optimal solution to the example instance

many crossings anyways (if such a situation is encountered in practice, one would switch to another way of representing the information); so, one can safely assume that the parameter is indeed small.

This is also true for other $\mathcal{NP}$-hard subproblems that relate to the Sugiyama approach. However, no easy solutions should be expected. For example, the DIRECTED FEEDBACK ARC SET PROBLEM [1] that is equivalent to the first phase is not known to admit a nice parameterized algorithm, see [2].

Experimental Results

Suderman [10] reports on experiments with nearly all problem variants discussed above, also see [11] for a better accessible presentation of some of the experimental results.

URL to Code

Suderman presents several JAVA applets related to the problems discussed in this article, see http://cgm.cs.mcgill.ca/~msuder/.

Cross References

Other parameterized search tree algorithms are explained in the contribution ► Vertex Cover Search Trees by Chen, Kanj, and Jia.

Recommended Reading

1. Chen, J., Liu, Y., Lu, S., O'Sullivan, B., Razgon, I.: A Fixed-Parameter Algorithm for the Directed Feedback Vertex Set Problem. In: 40th ACM Symposium on Theory of Computing STOC 2008, May 17–20, Victoria (BC), Canada (2008)
2. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, Berlin (1999)
3. Dujmović, V., Fellows, M.R., Hallett, M., Kitching, M., Liotta, G., McCartin, C., Nishimura, N., Ragde, P., Rosamond, F.A., Suderman, M., Whitesides, S., Wood, D.R.: A fixed-parameter approach to 2-layer planarization. Algorithmica **45**, 159–182 (2006)
4. Dujmović, V., Fernau, H., Kaufmann, M.: Fixed parameter algorithms for one-sided crossing minimization revisited. In: Liotta G. (ed.) Graph Drawing, 11th International Symposium GD 2003. LNCS, vol. 2912, pp. 332–344. Springer, Berlin (2004). A journal version has been accepted to J. Discret. Algorithms, see doi: 10.1016/j.jda.2006.12.008
5. Dujmović, V., Whitesides, S.: An efficient fixed parameter tractable algorithm for 1-sided crossing minimization. Algorithmica **40**, 15–32 (2004)
6. Eades, P., Wormald, N.C.: Edge crossings in drawings of bipartite graphs. Algorithmica **11**, 379–403 (1994)
7. Fernau, H.: Two-layer planarization: improving on parameterized algorithmics. J. Graph Algorithms Appl. **9**, 205–238 (2005)
8. Fernau, H., Kaufmann, M., Poths, M.: Comparing trees via crossing minimization. In: Ramanujam R., Sen S. (eds.) Foundations of Software Technology and Theoretical Computer Science FSTTCS 2005. LNCS, vol. 3821, pp. 457–469. Springer, Berlin (2005)
9. Nagamochi, H.: An improved bound on the one-sided minimum crossing number in two-layered drawings. Discret. Comput. Geom. **33**, 569–591 (2005)
10. Suderman, M.: Layered Graph Drawing. Ph.D. thesis, McGill University, Montréal (2005)
11. Suderman, M., Whitesides, S.: Experiments with the fixed-parameter approach for two-layer planarization. J. Graph Algorithms Appl. **9**(1), 149–163 (2005)
12. Sugiyama, K., Tagawa, S., Toda, M.: Methods for visual understanding of hierarchical system structures. IEEE Trans. Syst. Man Cybernet. **11**(2), 109–125 (1981)

Parameterized Matching

1993; Baker

MOSHE LEWENSTEIN
Department of Computer Science, Bar-Ilan University, Ramat-Gan, Israel

Problem Definition

Parameterized strings, or *p-strings*, are strings that contain both ordinary symbols from an alphabet Σ and parameter symbols from an alphabet Π. Two equal-length p-strings s and s' are a parameterized match, or *p-match*, if one p-string can be transformed into the other by applying a one-to-one function that renames the parameter symbols. The following example of a p-match is one with both ordinary and parameter symbols. The ordinary symbols are in lowercase and the parameter symbols are in uppercase.

$$s = A\ b\ A\ b\ C\ A\ d\ b\ A\ C\ d\ d$$
$$s' = D\ b\ D\ b\ E\ D\ d\ b\ D\ E\ d\ d$$

In some of the problems to be considered it will be sufficient to solve for p-strings in which all symbols are parameter symbols, as this is the more difficult part of the problem. In other words, the case in which $\Sigma = \emptyset$. In this case the definition can be reformulated so that s and s' are a p-match if there exists a bijection $\pi : \Pi_s \to \Pi_{s'}$, such that $\pi(s) = s'$, where $\pi(s)$ is the renaming of each character of s via π.

The following problems will be considered. *Parameterized matching* – given a parameterized pattern p of length m and parameterized text t, find all locations i of a parameterized text t for which p p-matches $t_i \cdots t_{i+m-1}$, where $m = |p|$. The same problem is also considered in two dimensions. *Approximate parameterized matching*– find all substrings of a parameterized text t that are approximate parameterized matches of a parameterized pattern p (to be fully defined later).

Key Results

Baker [4] introduced parameterized matching in the framework of her seminal work on discovering duplicate code within large programs for the sake of code minimization. An example of two code fragments that p-match taken from the X Windows system can be found in [4].

Parameterized Suffix Trees

In [4] and in the follow-up journal versions [6,7] a novel method was presented for parameterized matching by constructing *parameterized suffix trees*. The advantage of the parameterized suffix tree is that it supports indexing, i. e., one can preprocess a text and subsequently answer parameterized queries p in $O(|p|)$ time. In order to achieve parameterized suffix trees it is necessary to introduce the concept of a *predecessor string*. A *predecessor string* of a string s has at each location i the distance between i and the location containing the previous appearance of the symbol. The first appearance of each symbol is replaced with a 0. For example, the predecessor string of *aabbaba* is $0, 1, 0, 1, 3, 2, 2$. A simple and well-known fact is that:

Observation [7] s and s' p-match if and only if they have the same predecessor string.

Notice that this implies transitivity of parameterized matching, since if s and s' p-match and s' and s'' p-match then, by the observation, s and s' have the same predecessor string and, likewise, s' and s'' have the same predecessor string. This implies that s and s'' have the same predecessor string and hence, by the observation, p-match.

Moreover, one may also observe that if r is a prefix of s then the predecessor string of r, by definition, is exactly the $|r|$-length prefix of the predecessor string of s. Hence, similar to regular pattern matching, a parameterized pattern p p-matches at location i of t if and only if the $|p|$-length predecessor string of p is equal to the $|p|$-length prefix of the predecessor string of the suffix $t_i \cdots t_n$. Combining these observations it is natural to do as follows; create a (parameterized suffix) tree with a leaf for each suffix where the path from the root to the leaf corresponding to a given suffix will have its predecessor string labeling the path. Branching in the parameterized suffix tree, as with suffix trees, occurs according to the labels of the predecessor strings. See [4,6,7] for an example.

Baker's method essentially mimics the McCreight suffix tree construction [18]. However, while the suffix tree and the parameterized suffix tree are very similar, there is a slight hitch. A strong component of the suffix tree construction is the suffix link. This is used for the construction and, sometimes, for later pattern searches. The suffix link is based on the *distinct right context* property, which does not hold for the parameterized suffix tree. In fact, the node that is pointed to by the suffix link may not even exist. The main parts of [6,7] are dedicated to circumventing this problem.

In [7] Baker added the notion of "bad" suffix links, which point to the vertex just above, i. e., closer to the root than the desired place, and of updating them with a lazy evaluation when they are used. The algorithm runs in time $O(n|\Pi|\log|\Sigma|)$. In [6] (which is chronologically later than [7] despite being the first to appear) Baker changed the definition of "bad" suffix links to point to just below the desired place. This turns out to have nice properties and one can use more sophisticated data structures to improve the construction time to $O(n(|\Pi| + \log|\Sigma|))$.

Kosaraju [16] made a careful analysis of Baker's properties utilized in the algorithm of [6] which suffer from the $|\Pi|$ factor. He pointed out two sources for this large factor. He handled these two issues by using a concatenable queue and maintaining it in a lazy manner. This is sufficient to reduce the $|\Pi|$ factor to a $\log|\Pi|$ factor, yielding an algorithm of time $O(n(\log|\Pi| + \log|\Sigma|))$.

Obviously if the alphabet or symbol set is large the construction time may be $O(n \log n)$. Cole and Hariharan [9] showed how to construct the parameterized suffix trees in randomized $O(n)$ time for alphabets and parameters taken from a polynomially sized range, e. g., $[1, \cdots, n^c]$. They did this by adding additional nodes to the tree in a back-propagation manner which is reminiscent of fractional cascading. They showed that this adds only $O(n)$ nodes and allows the updating of the missing suffix links. However, this causes other problems and one may find the details of how this is handled in their paper.

More Methods for Parameterized Matching

Obviously the parameterized suffix tree efficiently solves the parameterized matching problem. Nevertheless, a couple of other results on parameterized matching are worth mentioning.

First, in [6] it was shown how to construct the parameterized suffix tree for the pattern and then to run the parameterized text through it, giving an algorithm with $O(m)$ space instead of $O(n)$.

Amir et al. [2] presented a simple method to solve the parameterized matching problem by mimicking the algorithm of Knuth, Morris and Pratt. Their algorithm works in $O(n * \min(\log|\Pi|, m))$ time independent of the alphabet size ($|\Sigma|$). Moreover, they proved that the log factor cannot be avoided for large symbol sets.

In [5] parameterized matching was solved with a Boyer–Moore type algorithm. In [10] the problem was solved with a Shift–Or type algorithm. Both handle the average case efficiently. In [10] emphasis was also put on the case of multiple parameterized matching, which was previously solved in [14] with an Aho–Corasick automaton-style algorithm.

Two-Dimensional Parameterized Matching

Two-dimensional parameterized matching arises in applications of image searching; see [13] for more details. Two-dimensional parameterized matching is the natural extension of parameterized matching where one seeks p-matches of a two-dimensional parameterized pattern p within a two-dimensional parameterized text t. It must be pointed out that classical methods for two-dimensional pattern matching, such as the L-suffix tree method, fail for parameterized matching. This is because known methods tend to cut the text and pattern into pieces to avoid going out of boundaries of the pattern. This is fine because each pattern piece can be individually evaluated (checked for equality) to a text piece. However, in parameterized matching there is a strong dependency between the pieces.

In [1] an innovative solution for the problem was given based on a collection of linearizations of the pattern and text with the property to be currently described. Consider a linearization. Two elements with the same character, say 'a,' in the pattern are defined to be neighbors if there is no other 'a' between them in this linearization. Now take all the 'a's of the pattern and create a graph G_a with 'a's as the nodes and edges between two if they are neighbors in some linearization. We say that two 'a's are chained if there is a path from one to the other in G_a. Applying one-dimensional parameterized matching on these linearizations ensures that any two elements that are chained will be evaluated to map to the same text value (the parameterized property). A collection of linearizations has the *fully chained* property if every two locations in p with the same character are chained. It was shown in [1] that one can obtain a collection of $\log m$ linearizations that is fully chained and that does not exceed pattern boundary limits. Each such linearization is solved with a convolution-based pattern matching algorithm. This takes $O(n^2 \log m)$ time for each linearization, where the text size is n^2. Hence, overall the time is $O(n^2 \log^2 m)$.

A different solution was proposed in [13], where it was shown that it is possible to solve the problem in $O(n^2 + m^{2.5}\text{polylog } m)$, where the text size is $O(n^2)$ and the pattern size is $O(m^2)$. Clearly, this is more efficient for large texts.

Approximate Parameterized Matching

Our last topic relates to parameterized matching in the presence of errors. Errors occur in the various applications and it is natural to consider parameterized matching with the Hamming distance metric or the edit distance metric.

In [8] the parameterized matching problem was considered in conjunction with the edit distance. Here the definition of edit distance was slightly modified so that the edit operations are defined to be insertion, deletion and parameterized replacements, i. e., the replacement of a substring with a string that p-matches it. An algorithm for finding the "parameterized edit distance" of two strings was devised whose efficiency is close to the efficiency of the algorithms for computing the classical edit distance.

However, it turns out that the operation of parameterized replacement relaxes the problem to an easier problem. The reason that the problem becomes easier is that two substrings that participate in two parameterized replacements are independent of each other (in the parameterized sense).

A more rigid, but more realistic, definition for the Hamming distance variant was given in [3]. For a pair of equal-length strings s and s' and a bijection π defined on the alphabet of s, the π-mismatch is the Hamming distance between the image under π of s and s'. The minimal π-mismatch over all bijections π is the approximate parameterized match. The problem considered in [3] is to find for each location i of a text t the approximate parameterized match of a pattern p with the substring beginning at location i. In [3] the problem was defined and linear-time algorithms were given for the case where the pattern is binary or the text is binary. However, this solution does not carry over to larger alphabets.

Unfortunately, under this definition the methods for classical string matching with errors for the Hamming distance, also known as pattern matching with mismatches, seem to fail. Following is an outline of a classical method [17] for pattern matching with mismatches that uses suffix trees.

The pattern is compared separately with each suffix of the text, beginning at locations $1 \le i \le n - m + 1$. Using a suffix tree of the text and precomputed longest common ancestor information (which can be computed once in linear time [11]), one can find the longest common prefix of the pattern and the corresponding suffix (in constant time). There must be a mismatch immediately afterwards. The algorithm jumps over the mismatch and repeats the process, taking into consideration the offsets of the pattern and suffix.

When attempting to apply this technique to a parameterized suffix tree, it fails. To illustrate this, consider the first matching substring (up until the first error) and the next matching substring (after the error). Both of these substrings p-match the substring of the text that they are aligned with. However, it is possible that combined they do not form a p-match. See the example below. In the example *abab* p-matches *cdcd* followed by a mismatch and subsequently followed by *abaa* p-matching *efee*. However, different π's are required for the local p-matches. This example also emphasizes why the definition of [8] is a simplification. Specifically, each local p-matching substring is one replacement, i. e., *abab* with *cdcd* is one replacement and *abaa* with *efee* is one more replacement. However, the definition of [3] captures the globality of the parameterized matching, not allowing, in this case, *abab* to p-match to two different substrings.

$$p = a\ b\ a\ b\ a\ a\ b\ a\ a\ \cdots$$
$$t = \cdots\ c\ d\ c\ d\ d\ e\ f\ e\ e\ \cdots$$

In [12] the problem of *parameterized matching with k mismatches* was considered. The parameterized matching problem with k mismatches seeks all locations i in text t where the minimal π-mismatch between p to $t_i \cdots t_{i+m-1}$ has at most k mismatches. An $O(nk^{1.5} + mk \log m)$ time algorithm was presented in [12]. At the base of the algorithm, i. e., for the case where $|p| = |t| = m$, an $O(m + k^{1.5})$ algorithm is used based on maximum matching algorithms. Then the algorithm uses a doubling scheme to handle the growing distance between potential parameterized matches (with at most k mismatches). Also shown in [12] is a strong relationship between maximum matching algorithms in sparse graphs and parameterized matching with k errors.

The rigid, but more realistic, definition for the Hamming distance version given in [3] can be naturally extended to the edit distance. Lately, it was shown that this problem is nondeterministic polynomial-time complete [15].

Applications

Parameterized matching has applications in code duplication detection in programming languages, in homework plagiarism detection, and in image processing, among others [1,4].

Cross References

- ► Multidimensional String Matching
- ► Sequential Approximate String Matching
- ► Sequential Exact String Matching
- ► Suffix Tree Construction in Hierarchical Memory
- ► Suffix Tree Construction in RAM

Recommended Reading

1. Amir, A., Aumann, Y., Cole, R., Lewenstein, M., Porat, E.: Function matching: Algorithms, applications and a lower bound. In: Proc. of the 30th International Colloquium on Automata, Languages and Programming (ICALP), 2003 pp. 929–942
2. Amir, A., Farach, M., Muthukrishnan, S.: Alphabet dependence in parameterized matching. Inf. Process. Lett. **49**, 111–115 (1994)
3. Apostolico, A., Erdös, P., Lewenstein, M.: Parameterized matching with mismatches. J. Discret. Algorithms **5**(1), 135–140 (2007)
4. Baker, B.S.: A theory of parameterized pattern matching: algorithms and applications. In: Proc. 25th Annual ACM Symposium on the Theory of Computation (STOC), 1993, pp. 71–80
5. Baker, B.S.: Parameterized pattern matching by Boyer-Moore-type algorithms. In: Proc. 6th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 1995, pp. 541–550
6. Baker, B.S.: Parameterized pattern matching: Algorithms and applications. J. Comput. Syst. Sci. **52**(1), 28–42 (1996)
7. Baker, B.S.: Parameterized duplication in strings: Algorithms and an application to software maintenance. SIAM J. Comput. **26**(5), 1343–1362 (1997)
8. Baker, B.S.: Parameterized diff. In: Proc. 10th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 1999, pp. 854–855
9. Cole, R., Hariharan, R.: Faster suffix tree construction with missing suffix links. In: Proc. 32nd ACM Symposium on Theory of Computing (STOC), 2000 pp. 407–415
10. Fredriksson, K., Mozgovoy, M.: Efficient parameterized string matching. Inf. Process. Lett. **100**(3), 91–96 (2006)
11. Harel, D., Tarjan, R.E.: Fast algorithms for finding nearest common ancestor. J. Comput. Syst. Sci. **13**, 338–355 (1984)
12. Hazay, C., Lewenstein, M., Sokol, D.: Approximate parameterized matching. ACM Trans. Algorithms **3**(3) (2007)
13. Hazay, C., Lewenstein, M., Tsur, D.: Two dimensional parameterized matching. In: Proc. of 16th Symposium on Combinatorial Pattern Matching (CPM), 2005, pp. 266–279
14. Idury, R.M., Schäffer, A.A.: Multiple matching of parametrized patterns. Theor. Comput. Sci. **154**(2), 203–224 (1996)
15. Keller, O., Kopelowitz, T., Lewenstein, M.: Parameterized LCS and edit distance are NP-Complete. Manuscript
16. Kosaraju, S.R.: Faster algorithms for the construction of parameterized suffix trees. In: Proc. 36th Annual Symposium on Foundations of Computer Science (FOCS), 1995, pp. 631–637
17. Landau, G.M., Vishkin, U.: Fast string matching with k differences. J. Comput. Syst. Sci. **37**(1), 63–78 (1988)
18. McCreight, E.M.: A space-economical suffix tree construction algorithm. J. ACM **23**, 262–272 (1976)

Parameterized SAT

2003; Szeider

STEFAN SZEIDER
Department of Computer Science, Durham University, Durham, UK

Keywords and Synonyms

Structural parameters for SAT

Problem Definition

Much research has been devoted to finding classes of propositional formulas in conjunctive normal form (CNF) for which the recognition problem as well as the propositional satisfiability problem (SAT) can be decided in polynomial time. Some of these classes form infinite chains $C_1 \subset C_2 \subset \cdots$ such that every CNF formula is contained in some C_k for k sufficiently large. Such classes are typically of the form $C_k = \{F \in \text{CNF}\colon \pi(F) \leq k\}$, where π is a computable mapping that assigns to CNF formulas F a non-negative integer $\pi(F)$; we call such a mapping a *satisfiability parameter*. Since SAT is an NP-complete problem (actually, the first problem shown to be NP-complete [1]), we must expect that, the larger k, the longer the worst-case running times of the polynomial-time algorithms that recognize and decide satisfiability of formulas in C_k. Whence there is a certain tradeoff between the generality of classes and the performance guarantee for the corresponding algorithms. Szeider [12] initiates a broad investigation of this tradeoff in the conceptional framework of *parameterized complexity* [2,3,6]. This investigation draws attention to satisfiability parameters π for which the following holds: recognition and satisfiability decision of formulas F with $\pi(F) \leq k$ can be carried out in *uniform polynomial time*, that is, by algorithms with running time bounded by a polynomial whose order is independent of k (albeit, possibly involving a constant factor that is exponential in k). If a satisfiability parameter π allows satisfiability decision in uniform polynomial time, we say that *SAT is fixed-parameter tractable with respect to* π.

Satisfiability Parameters Based on Graph Invariants

One can define satisfiability parameters by means of certain graphs associated with CNF formulas. The *primal graph* of a CNF formula is the graph whose vertices are the variables of the formula; two variables are joined by an edge if the variables occur together in a clause. The *incidence graph* of a CNF formula is the bipartite graph whose vertices are the variables and clauses of the formula; a variable and a clause are joined by an edge if the variable occurs in the clause.

Satisfiability Parameters Based on Backdoor Sets

The concept of backdoor sets [13] gives rise to several interesting satisfiability parameters. Let C be a class of CNF formulas. A set B of variables of a CNF formula F is a *strong* C*-backdoor set* if for every partial truth assignment $\tau\colon B \to \{\text{true}, \text{false}\}$, the restriction of F to τ belongs to C. Here, the restriction of F to τ is the CNF formula obtained from F by removing all clauses that contain a literal that is true under τ and by removing from the remaining clauses all literals that are false under τ.

Key Results

Theorem 1 (Gottlob, Scarcello, and Sideri [4]) *SAT is fixed-parameter tractable with respect to the treewidth of primal graphs.*

Several satisfiability parameters that generalize the treewidth of primal graphs, such as the *treewidth and clique-width of incidence graphs*, have been studied [5,10,12].

The *maximum deficiency* of a CNF formula F is the number of clauses remaining exposed by a maximum matching of the incidence graph of F.

Theorem 2 (Szeider [11]) *SAT is fixed-parameter tractable with respect to maximum deficiency.*

A CNF formula is *minimal unsatisfiable* if it is unsatisfiable but removing any of its clauses makes it satisfiable. Recognition of minimal unsatisfiable formulas is DP-complete [9].

Corollary 1 (Szeider [11]) *Recognition of minimal unsatisfiable CNF formulas is fixed-parameter tractable with respect to the difference between the number of clauses and the number of variables.*

Theorem 3 (Nishimura, Ragde, and Szeider [7]) *SAT is fixed-parameter tractable with respect to the size of strong* HORN*-backdoor sets and with respect to the size of strong* 2CNF*-backdoor sets.*

Applications

Satisfiability provides a powerful and general formalism for solving various important problems including hardware and software verification and planning. Instances stemming from applications usually contain a "hidden structure" (see, e. g. [13]). The satisfiability parameters considered above are designed to make this hidden structure explicit in the form of small values for the parameter. Thus, satisfiability parameters are a way to make the hidden structure accessible to an algorithm.

Open Problems

A new line of research is concerned with the identification of further satisfiability parameters that allow a fixed-parameter tractable SAT decision are more general than the known parameters and apply well to real-world problem instances.

Cross References

► Maximum Matching
► Treewidth of Graphs

Recommended Reading

1. Cook, S.A.: The complexity of theorem-proving procedures. In: Proc. 3rd Annual Symp. on Theory of Computing, Shaker Heights, OH 1971, pp. 151–158
2. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Monographs in Computer Science. Springer, Berlin (1999)
3. Flum, J., Grohe, M.: Parameterized Complexity Theory. Texts in Theoretical Computer Science, vol. XIV. An EATCS Series. Springer, Berlin (2006)
4. Gottlob, G., Scarcello, F., Sideri, M.: Fixed-parameter complexity in AI and nonmonotonic reasoning. Artif. Intell. **138**, 55–86 (2002)
5. Gottlob, G., Szeider, S.: Fixed-parameter algorithms for artificial intelligence, constraint satisfaction, and database problems. Comput. J., Special Issue on Parameterized Complexity, Advanced Access (2007)
6. Niedermeier, R.: Invitation to Fixed-Parameter Algorithms, Oxford Lecture Series in Mathematics and Its Applications. Oxford University Press, Oxford, UK (2006)
7. Nishimura, N., Ragde, P., Szeider, S.: Detecting backdoor sets with respect to Horn and binary clauses. In: Informal proceedings of SAT 2004, 7th International Conference on Theory and Applications of Satisfiability Testing, Vancouver, BC, Canada, 10–13 May 2004, pp. 96–103
8. Nishimura, N., Ragde, P., Szeider, S.: Solving SAT using vertex covers. Acta Inf. **44**(7–8), 509–523 (2007)
9. Papadimitriou, C.H., Wolfe, D.: The complexity of facets resolved. J. Comput. Syst. Sci. **37**, 2–13 (1988)
10. Samer, M., Szeider, S.: Algorithms for propositional model counting. In: Proceedings of LPAR 2007, 14th International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Yerevan, Armenia, 15–19 October 2007. Lecture Notes in Computer Science, vol. 4790, pp. 484–498. Springer, Berlin (2007)
11. Szeider, S.: Minimal unsatisfiable formulas with bounded clause-variable difference are fixed-parameter tractable. J. Comput. Syst. Sci. **69**, 656–674 (2004)
12. Szeider, S.: On fixed-parameter tractable parameterizations of SAT. In: Giunchiglia, E., Tacchella, A. (eds.) Theory and Applications of Satisfiability, 6th International Conference, SAT 2003, Selected and Revised Papers. Lecture Notes in Computer Science, vol. 2919, pp. 188–202. Springer, Berlin (2004)
13. Williams, R., Gomes, C., Selman, B.: On the connections between backdoors, restarts, and heavy-tailedness in combinatorial search, In: informal proceedings of SAT 2003 (Sixth International Conference on Theory and Applications of Satisfiability Testing, 5–8 May 2003, S. Margherita Ligure – Portofino, Italy), 2003, pp. 222–230

Pattern Matching

► Multidimensional Compressed Pattern Matching
► Two Dimensional Scaled Pattern Matching

Peer to Peer

► P2P

Peptide De Novo Sequencing with MS/MS

2005; Ma, Zhang, Liang

Bin Ma
Department of Computer Science, University of Western Ontario, London, ON, Canada

Keywords and Synonyms

De novo sequencing; Peptide sequencing

Problem Definition

De novo sequencing arises from the identification of peptides by using tandem mass spectrometry (MS/MS). A peptide is a sequence of amino acids in biochemistry and can be regarded as a string over a finite alphabet from a computer scientist's point of view. Each letter in the alphabet represents a different kind of amino acid, and is associate with a mass value. In the biochemical experiment, a tandem mass spectrometer is utilized to fragment many copies of the peptide into pieces, and to measure the mass values (in fact, the mass to charge ratios) of the fragments simultaneously. This gives a tandem mass spectrum. Since different peptides normally produce different spectra, it is possible, and now a common practice, to deduce the amino acid sequence of the peptide from its spectrum. Often this deduction involves the searching in a database for a peptide that can possibly produce the spectrum. But in many cases such a database does not exist or is not complete, and the calculation has to be done without looking for a database. The latter approach is called *de novo* sequencing.

A general form of *de novo* sequencing problems is described in [2]. First, a score function $f(P, S)$ is defined to evaluate the pairing of a peptide P and a spectrum S. Then the *de novo* sequencing problem seeks for a peptide P such that $f(P, S)$ is maximized for a given spectrum S.

When the peptide is fragmented in the tandem mass spectrometer, many types of fragments can be generated. The most common fragments are the so called b-ions and y-ions. B-ions correspond to the prefixes of the peptide sequence, and y-ions the suffixes. Readers are referred to [8] for the biochemical details of the MS/MS experiments and the possible types of fragment ions. For clarity, in what follows only b-ions and y-ions are considered, and the *de novo* sequencing problem will be formulated as a pure computational problem.

A spectrum $S = \{(x_i, h_i)\}$ is a set of peaks, each has a mass value x_i and an intensity value h_i. A peptide $P = a_1 a_2 \dots a_n$ is a string over a finite alphabet Σ. Each $a \in \Sigma$ is associated with a positive mass value $m(a)$. For any string $t = t_1 t_2 \dots t_k$, denote $m(t) = \sum_{i=1}^{k} m(t_i)$. The mass of a length-$i$ prefix (b-ion) of P is defined as $b_i = c_b + m(a_1 a_2 \dots a_i)$. The mass of a length-$i$ suffix (y-ion) of P is defined as $y_i = c_y + m(a_{k-i+1} \dots a_{k-1} a_k)$. Here c_b and c_y are two constants related to the nature of the MS/MS experiments. If the mass unit used for measuring each amino acid is dalton, then $c_b = 1$ and $c_y = 19$.

Let δ be a mass error tolerance that is associated with the mass spectrometer. For mass value m, the peaks *matched* by m is defined as $D(m) = \{(x_i, h_i) \in S \mid |x_i - m| \le \delta\}$. The general idea of *de novo* sequencing is to maximize the number and intensities of the peaks matched by all b and y ions. Normally, δ is far less than the minimum mass of an amino acid. Therefore, for different i and j, $D(b_i) \cap D(b_j) = \emptyset$ and $D(y_i) \cap D(y_j) = \emptyset$. However, $D(b_i)$ and $D(y_j)$ may share common peaks. So, if not defined carefully, a peak may be counted twice in the score function. There are two different definitions of *de novo* sequencing problem, corresponding to two different ways of handling this situation.

Definition 1 (Anti-symmetric de novo sequencing)
Instance: A spectrum S, a mass value M, and an error tolerance δ.
Solution: A peptide P such that $m(P) = M$, and $D(b_i) \cap D(y_j) = \emptyset$ for any i, j.
Objective: Maximize $\sum_{k=1}^{n} \sum_{(x_i,h_i)\in D(b_k)\cup D(y_k)} h_i$.

This definition discards the peptides that gives a pair of b_i and y_j with similar mass values, because this happens rather infrequently in practice. Another definition allows the peptides to have pairs of b_i and y_j with similar mass values. However, when a peak is matched by multiple ions, it is counted only once. More precisely, define the matched peaks by P as

$$D(P) = \bigcup_{i=1}^{n} (D(b_i) \cup D(y_i)) \,.$$

Definition 2 (De novo sequencing)
Instance: A spectrum S, a mass value M, and an error tolerance δ.
Solution: A peptide P such that $m(P) = M$.
Objective: Maximize $f(P, S) = \sum_{(x_i,h_i)\in D(P)} h_i$.

Key Results

Anti-symmetric de novo sequencing was studied in [1,2]. These studies convert the spectrum into a spectrum graph. Each peak in the spectrum generates a few of nodes in the spectrum graph, corresponding to the different types of ions that may produce the peak. Each edge in the graph indicates that the mass difference of the two adjacent nodes is approximately the mass of an amino acid, and the edge is labeled with the amino acid. When at least one of each pair of b_i and y_{n-i} matches a peak in the spectrum, the *de novo* sequencing problem is reduced to the finding of the "anti-symmetric" longest path in the graph. A dynamic programming algorithm for such purpose was published in [1].

Theorem 1 ([1]) *The longest anti-symmetric path in a spectrum graph $G = \langle V, E\rangle$ can be found in $O(|V||E|)$ time.*

Under definition 2, *de novo* sequencing was studied in [5] and a polynomial time algorithm was provided. The algorithm is again a dynamic programming algorithm. For two mass values (m, m'), the dynamic programming calculates an optimal pair of prefix Aa and suffix $a'A'$, such that
1. $m(Aa) = m$ and $m(a'A') = m'$.
2. Either $c_b + m(A) < c_y + m(a'A') \le c_b + m(Aa)$ or $c_y + m(A') \le c_b + m(A) < c_y + m(a'A')$.

The calculation for (m, m') is based on the optimal solutions of smaller mass values. Because of the second above requirement, it is proved in [5] that not all pairs of (m, m') are needed. This is used to speed up the algorithm. A carefully designed strategy can eventually output a prefix and a suffix so that their concatenation form the optimal solution of the *de novo* sequencing problem. More specifically, the following theorem holds.

Theorem 2 ([6]) *The de novo sequencing problem has an algorithm that gives the optimal peptide in $O(|\Sigma| \times \delta \times \max_{a\in\Sigma} m(a) \times M)$.*

Because $|\Sigma|$, δ, $\max_{a \in \Sigma} m(a)$ are all constants, the algorithm in fact runs in linear time with a large coefficient.

Although the above algorithms are designed to maximize the total intensities of the matched peaks, they can be adapted to work on more sophisticated score functions. Some studies of other score functions can be found in [2,3,4,6]. Some of these score functions require new algorithms.

Applications

The algorithms have been implemented into software programs to assist the analyses of tandem mass spectrometry data. Software using the spectrum graph approach includes Sherenga [2]. The *de novo* sequencing algorithm under the second definition was implemented in PEAKS [6]. More complete lists of the *de novo* sequencing software and their comparisons can be found in [7,9].

URL to Code

PEAKS free trial version is available at http://www.bioinfor.com/.

Recommended Reading

1. Chen, T., Kao, M.-Y., Tepel, M., Rush J., Church, G.: A dynamic programming approach to de novo peptide sequencing via tandem mass spectrometry. J. Comput. Biol. **8**(3), 325–337 (2001)
2. Dančík, V., Addona, T., Clauser, K., Vath, J., Pevzner, P.: De novo protein sequencing via tandem mass-spectrometry. J. Comput. Biol. **6**, 327–341 (1999)
3. Fischer, B., Roth, V., Roos, F., Grossmann, J., Baginsky, S., Widmayer, P., Gruissem, W., Buhmann J.: NovoHMM: A Hidden Markov Model for de novo peptide sequencing. Anal. Chem. **77**, 7265–7273 (2005)
4. Frank, A., Pevzner, P.: Pepnovo: De novo peptide sequencing via probabilistic network modeling. Anal. Chem. **77**, 964–973 (2005)
5. Ma, B., Zhang, K., Liang, C.: An effective algorithm for the peptide de novo sequencing from MS/MS spectrum. J. Comput. Syst. Sci. **70**, 418–430 (2005)
6. Ma, B., Zhang, K., Lajoie, G., Doherty-Kirby, A., Hendrie, C., Liang, C., Li, M.: PEAKS: Powerful software for peptide de novo sequencing by tandem mass spectrometry. Rapid Commun. Mass Spectrom. **17**(20), 2337–2342 (2003)
7. Pevtsov, S., Fedulova, I., Mirzaei, H., Buck, C., Zhang, X.: Performance evaluation of existing de novo sequencing algorithms. J. Proteome Res. **5**(11), 3018–3028 (2006) ASAP Article 10.1021/pr060222h
8. Steen, H., Mann, M.: The ABC's (and XYZ's) of peptide sequencing. Nat. Rev. Mol. Cell Biol. **5**(9), 699–711 (2004)
9. Xu, C., Ma, B.: Software for Computational Peptide Identification from MS/MS. Drug Discov. Today **11**(13/14), 595–600 (2006)

Perceptron Algorithm

1959; Rosenblatt

SHAI SHALEV-SHWARTZ
Toyota Technological Institute, Chicago, IL, USA

Keywords and Synonyms

Online learning, Single layer neural network

Problem Definition

The Perceptron algorithm [1,13] is an iterative algorithm for learning classification functions. The Perceptron was mainly studied in the online learning model. As an online learning algorithm, the Perceptron observes instances in a sequence of trials. The observation at trial t is denoted by $\mathbf{x}_t$. After each observation, the Perceptron predicts a yes/no (+/−) outcome, denoted $\hat{y}_t$, which is calculated as follows

$$\hat{y}_t = \text{sign}(\langle \mathbf{w}_t, \mathbf{x}_t \rangle) ,$$

where $\mathbf{w}_t$ is a weight vector which is learned by the Perceptron and $\langle \cdot, \cdot \rangle$ is the inner product operation. Once the Perceptron has made a prediction, it receives the correct outcome, denoted y_t, where $y_t \in \{+1, -1\}$. If the prediction of the Perceptron was incorrect it updates its weight vector, presumably improving the chance of making an accurate prediction on subsequent trials. The update rule of the Perceptron is

$$\mathbf{w}_{t+1} = \begin{cases} \mathbf{w}_t + y_t \mathbf{x}_t & \text{if} \hat{y}_t \neq y_t \\ \mathbf{w}_t & \text{otherwise} \end{cases} . \qquad (1)$$

The quality of an online learning algorithm is measured by the number of prediction mistakes it makes along its run. Novikoff [12] and Block [2] have shown that whenever the Perceptron is presented with a sequence of linearly separable examples, it makes a bounded number of prediction mistakes which does not depend on the length of the sequence of examples. Formally, let $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_T, y_T)$ be a sequence of instance-label pairs. Assume that there exists a unit vector $\mathbf{u}$ ($\|\mathbf{u}\|_2 = 1$) and a positive scalar $\gamma > 0$ such that for all t, $y_t(\mathbf{u} \cdot \mathbf{x}_t) \geq \gamma$. In words, $\mathbf{u}$ separates the instance space into two half-spaces such that positively labeled instances reside in one half-space while the negatively labeled instances belong to the second half-space. Moreover, the distance of each instance to the separating hyperplane $\{\mathbf{x} : \mathbf{u} \cdot \mathbf{x} = 0\}$, is at least γ. The scalar γ is often referred to as the margin attained by $\mathbf{u}$ on the sequence of examples. Novikoff and Block proved that the number of prediction mistakes the Perceptron makes on a sequence of linearly separable examples is at most $(R/\gamma)^2$, where $R = \max_t \|\mathbf{x}_t\|_2$ is the minimal radius of a ball enclosing all the instances. In 1969, Minsky and Papert [11] underscored serious limitations of the Perceptron by showing that it is impossible to learn many classes of patterns using the Perceptron (for example, XOR functions). This fact caused a significant decrease of interest in the Perceptron. The Perceptron has gained back its popularity after Freund and Schapire [9] proposed to use it in conjunction with kernels. The kernel-based Perceptron not only can handle non-separable datasets but can also be utilized for efficiently classifying non-vectorial instances such as trees and strings (see for example [5]).

To implement the Perceptron in conjunction with kernels one can utilize the fact that at each trial of the algorithm, the weight vector $\mathbf{w}_t$ can be written as a linear combination of the instances

$$\mathbf{w}_t = \sum_{i \in I_t} y_i \, \mathbf{x}_i \, ,$$

where $I_t = \{i < t : \hat{y}_i \neq y_i\}$ is the indices of trials in which the Perceptron made a prediction mistake. Therefore, the prediction of the algorithm can be rewritten as

$$\hat{y}_t = \text{sign}\left(\sum_{i \in I_t} y_i \, \langle \mathbf{x}_i, \mathbf{x}_t \rangle \right) ,$$

and the update rule of the weight vector can be replaced with an update rule for the set of erroneous trials

$$I_{t+1} = \begin{cases} I_t \cup \{t\} & \text{if } \hat{y}_t \neq y_t \\ I_t & \text{otherwise} \end{cases} . \tag{2}$$

In the kernel-based Perceptron, the inner product $\langle \mathbf{x}_i, \mathbf{x}_t \rangle$ is replaced with a Mercer kernel function, $K(\mathbf{x}_i, \mathbf{x}_t)$, without any further changes to the algorithm (for a discussion on Mercer kernels see for example [15]). Intuitively, the kernel function $K(\mathbf{x}_i, \mathbf{x}_t)$ implements an inner product $\langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_t) \rangle$ where ϕ is a non-linear mapping from the original instance space into another (possibly high dimensional) Hilbert space. Even if the original instances are not linearly separable, the images of the instances due to the non-linear mapping ϕ can be linearly separable and thus the kernel-based Perceptron can handle non-separable datasets. Since the analysis of the Perceptron does not depend on the dimensionality of the instances, all of the formal results still hold when the algorithm is used in conjunction with kernel functions.

Key Results

In the following a mistake bound for the Perceptron in the non-separable case (see for example [10,14]) is provided.

Theorem *Assume that the Perceptron is presented with the sequence of examples $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_T, y_T)$ and denote $R = \max_t \|\mathbf{x}_t\|_2$. Let $\mathbf{u}$ be a unit length weight vector ($\|\mathbf{u}\|_2 = 1$), let $\gamma > 0$ be a scalar, and denote*

$$L = \sum_{t=1}^{T} \max\{0, 1 - y_t \langle \mathbf{u}/\gamma, \mathbf{x}_t \rangle\} .$$

Then, the number of prediction mistakes the Perceptron makes on the sequence of example is at most

$$L + \left(\frac{R}{\gamma}\right)^2 + \frac{R\sqrt{L}}{\gamma} .$$

Note that if there exists $\mathbf{u}$ and γ such that $y_t \langle \mathbf{u}, \mathbf{x}_t \rangle \geq \gamma$ for all t then $L = 0$ and the above bound reduces to Novikoff's bound,

$$\left(\frac{R}{\gamma}\right)^2 .$$

Note also that the bound does not depend on the dimensionality of the instances. Therefore, it holds for the kernel-based Perceptron as well with $R = \max_t K(\mathbf{x}_t, \mathbf{x}_t)$.

Applications

So far the Perceptron has been viewed in the prism of online learning. Freund and Schapire [9] proposed a simple conversion of the Perceptron algorithm to the batch learning setting. A batch learning algorithm receives as input a training set of examples $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_T, y_T)\}$ sampled independently from an underlying joint distribution over the instance and label space. The algorithm is required to output a single classification function which performs well on unseen examples as long as the unseen examples are sampled from the same distribution as the training set. The conversion of the Perceptron to the batch setting proposed by Freund and Schapire is called the voted Perceptron algorithm. The idea is to simply run the online Perceptron on the training set of examples, thus producing a sequence of weight vectors $\mathbf{w}_1, \ldots, \mathbf{w}_T$. Then, the single classification function to be used for unseen examples is a majority vote over the predictions of the weight vectors. That is,

$$f(\mathbf{x}) = \begin{cases} +1 & \text{if } |\{t : \langle \mathbf{w}_t, \mathbf{x} \rangle > 0\}| > |\{t : \langle \mathbf{w}_t, \mathbf{x} \rangle < 0\}| \\ -1 & \text{otherwise} \end{cases}$$

It was shown (see again [9]) that if the number of prediction mistakes the Perceptron makes on the training set is

Perceptron Algorithm, Table 1

Online Perceptron	Kernel-based Online Perceptron
INITIALIZATION: $\mathbf{w}_1 = \mathbf{0}$	INITIALIZATION: $\mathbf{I}_1 = \{\cdot\}$
For $t = 1, 2, \ldots$	For $t = 1, 2, \ldots$
Receive an instance $\mathbf{x}_t$	Receive an instance $\mathbf{x}_t$
Predict an outcome $\hat{y}_t = \text{sign}(\langle \mathbf{w}_t, \mathbf{x}_t \rangle)$	Predict an outcome $\hat{y}_t = \text{sign}\left(\sum_{i \in I_t} K(\mathbf{x}_i, \mathbf{x}_t)\right)$
Receive correct outcome $y_t \in \{+1, -1\}$	Receive correct outcome $y_t \in \{+1, -1\}$
Update: $\mathbf{w}_{t+1} = \begin{cases} \mathbf{w}_t + y_t \mathbf{x}_t & \text{if } \hat{y}_t \neq y_t \\ \mathbf{w}_t & \text{otherwise} \end{cases}$	Update: $I_{t+1} = \begin{cases} I_t \cup \{t\} & \text{if } \hat{y}_t \neq y_t \\ I_t & \text{otherwise} \end{cases}$

small, then $f(\mathbf{x})$ is likely to perform well on unseen examples as well.

Finally, it should be noted that the Perceptron algorithm was used for other purposes such as solving linear programming [3] and training support vector machines [14]. In addition, variants of the Perceptron was used for numerous additional problems such as online learning on a budget [8,4], multiclass categorization and ranking problems [6,7], and discriminative training for hidden Markov models [5].

Cross References

► Support Vector Machines

Recommended Reading

1. Agmon., S.: The relaxation method for linear inequalities. Can. J. Math. **6**(3), 382–392 (1954)
2. Block., H. D.: The perceptron: A model for brain functioning. Rev. Mod. Phys. **34**, 123–135 (1962)
3. Blum, A., Dunagan J. D.: Smoothed analysis of the perceptron algorithm for linear programming. In: SODA, (2002)
4. Cesa-Bianchi, N., Gentile, C.: Tracking the best hyperplane with a simple budget perceptron. In: Proceedings of the Nineteenth Annual Conference on Computational Learning Theory, (2006)
5. Collins, M.: Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In: Conference on Empirical Methods in Natural Language Processing, (2002)
6. Crammer, K., Dekel, O., Keshet, J., Shalev-Shwartz, S., Singer, Y.: Online passive aggressive algorithms. J. Mach. Learn. Res. **7** (2006)
7. Crammer, K., Singer, Y.: A new family of online algorithms for category ranking. In: Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (2002)
8. Dekel, O., Shalev-Shwartz, S., Singer, Y.: The Forgetron: A kernel-based perceptron on a fixed budget. In: Advances in Neural Information Processing Systems 18 (2005)
9. Freund, Y., Schapire, R. E.: Large margin classification using the perceptron algorithm. In: Proceedings of the Eleventh Annual Conference on Computational Learning Theory (1998)
10. Gentile, C.: The robustness of the p-norm algorithms. Mach. Learn. **53**(3) (2002)
11. Minsky, M., Papert, S.: Perceptrons: An Introduction to Computational Geometry. The MIT Press, (1969)
12. Novikoff, A. B. J.: On convergence proofs on perceptrons. In: Proceedings of the Symposium on the Mathematical Theory of Automata, volume XII, pp. 615–622, (1962)
13. Rosenblatt, F.: The perceptron: A probabilistic model for information storage and organization in the brain. Psychol. Rev. **65**, 386–407 (1958)
14. Shalev-Shwartz, S., Singer, Y.: A new perspective on an old perceptron algorithm. In: Proceedings of the Eighteenth Annual Conference on Computational Learning Theory, (2005)
15. Vapnik, V. N.: Statistical Learning Theory. Wiley (1998)

Perfect Phylogeny (Bounded Number of States)

1997; Kannan, Warnow

JESPER JANSSON
Ochanomizu University, Tokyo, Japan

Keywords and Synonyms

Compatibility of characters with a bounded number of states; Convex tree-realization of partitions containing a bounded number of sets

Problem Definition

Let $S = \{s_1, s_2, \ldots, s_n\}$ be a set of elements called *objects* and *species*, and let $C = \{c_1, c_2, \ldots, c_m\}$ be a set of functions called *characters* such that each $c_j \in C$ is a function from S to the set $\{0, 1, \ldots, r_j - 1\}$ for some integer r_j. For every $c_j \in C$, the set $\{0, 1, \ldots, r_j - 1\}$ is called the set of *allowed states* of character c_j, and for any $s_i \in S$ and $c_j \in C$, it is said that the *state of s_i on c_j* is α, or that the *state of c_j for s_i* is α, where $\alpha = c_j(s_i)$. The *character state matrix* for S and C is the $(n \times m)$-matrix in which entry (i, j) for any

$i \in \{1, 2, \ldots, n\}$ and $j \in \{1, 2, \ldots, m\}$ equals the state of s_i on c_j.

In this chapter, a *phylogeny for S* is an unrooted tree whose leaves are bijectively labeled by S. For every $c_j \in C$ and $\alpha \in \{0, 1, \ldots, r_j - 1\}$, define the set $S_{c_j,\alpha}$ by $S_{c_j,\alpha} = \{s_i \in S :$ the state of s_i on c_j is $\alpha\}$. A *perfect phylogeny for* (S, C) (if one exists) is a phylogeny T for S such that the following holds: for each $c_j \in C$ and pair of allowed states α, β of c_j with $\alpha \neq \beta$, the minimal subtree of T that connects $S_{c_j,\alpha}$ and the minimal subtree of T that connects $S_{c_j,\beta}$ are vertex-disjoint. See Fig. 1 for an example. *The Perfect Phylogeny Problem* is the following:

Problem 1 (The Perfect Phylogeny Problem)
INPUT: *A character state matrix M for some S and C.*
OUTPUT: *A perfect phylogeny for (S, C), if one exists; otherwise, null.*

Below, define $r = \max_{j \in \{1,2,\ldots,m\}} r_j$.

Key Results

The following negative result was proved by Bodlaender, Fellows, and Warnow [2] and, independently, by Steel [13]:

Theorem 1 ([2,13]) *The Perfect Phylogeny Problem is NP-hard.*

On the other hand, certain restrictions of The Perfect Phylogeny Problem can be solved efficiently. One important special case occurs if the number of allowed states of each character is limited[1]. For this case, Agarwala and Fernández-Baca [1] designed a dynamic programming-based algorithm that builds perfect phylogenies on certain subsets of S called *c-clusters* (also referred to as *proper clusters* in [5,10] and *character subfamilies* in [6]) in a bottom-up fashion. Each c-cluster G has the property that: (1) G and $S \setminus G$ share at most one state of each character; and (2) for at least one character, G and $S \setminus G$ share no states. The number of c-clusters is at most $2^r m$, and the algorithm's total running time is $O(2^{3r}(nm^3 + m^4))$, i. e., exponential in r. (Hence, The Perfect Phylogeny Problem is polynomial-time solvable if the number of allowed states of every character is upper-bounded by $O(\log(m + n))$.) Subsequently, Kannan and Warnow [10] presented a modified algorithm with improved running time. They restructured the algorithm of [1] to eliminate one of the three nested loops that steps through all possible c-clusters and added a pre-processing step which speeds up the innermost loop. The resulting time complexity is given by:

[1]For other variants of The Perfect Phylogeny Problem which can be solved efficiently, see, for example, entries ► Directed Perfect Phylogeny (Binary Characters) of this Encyclopedia or the survey by Fernández-Baca [5] .

Perfect Phylogeny (Bounded Number of States), Table 1
The running times of the fastest known algorithms for The Perfect Phylogeny Problem with a bounded number of states

r	Running time	Reference
2	$O(nm)$	[11] together with [7]
3	$\min\{O(nm^2), O(n^2m)\}$	[3,10] together with [9]
4	$\min\{O(nm^2), O(n^2m)\}$	[10] together with [9]
≥ 5	$O(2^{2r}nm^2)$	[10]

Theorem 2 ([10]) *The algorithm of Kannan and Warnow in [10] solves The Perfect Phylogeny Problem in $O(2^{2r}nm^2)$ time.*

A perfect phylogeny T for (S, C) is called *minimal* if no tree which results by contracting an edge of T is a perfect phylogeny for (S, C). In [10], Kannan and Warnow also showed how to extend their algorithm to enumerate all minimal perfect phylogenies for (S, C) by constructing a directed acyclic graph that implicitly stores the set of all perfect phylogenies for (S, C).

Theorem 3 ([10]) *The extended algorithm of Kannan and Warnow in [10] enumerates the set of all minimal perfect phylogenies for (S, C) so that the maximum computation time between two consecutive outputs is $O(2^{2r}nm^2)$.*

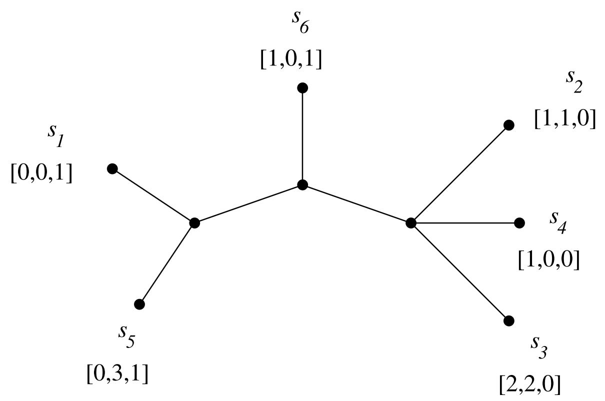
For very small values of r, even faster algorithms are known. Refer to the table in Table 1 for a summary. If $r = 2$ then the problem can be solved in $O(nm)$ time by reducing it to The *Directed* Perfect Phylogeny Problem for Binary Characters (see, e. g., Encyclopedia ► Directed Perfect Phylogeny (Binary Characters) for a definition of this variant of the problem) using $O(nm)$ time [7,11] and then applying Gusfield's $O(nm)$-time algorithm [7]. If $r = 3$ or $r = 4$, the problem is solvable in $O(n^2m)$ time by another algorithm by Kannan and Warnow [9], which is faster than the algorithm from Theorem 2 when $n < m$. Also note that for the case $r = 3$, there exists an older algorithm by Dress and Steel [3] whose running time coincides with that of the algorithm in Theorem 2.

Applications

A central goal in computational evolutionary biology and phylogenetic reconstruction is to develop efficient methods for constructing, from some given data, a phylogenetic tree that accurately describes the evolutionary relationships among a set of objects (e. g., biological species or

M	c_1	c_2	c_3
s_1	0	0	1
s_2	1	1	0
s_3	2	2	0
s_4	1	0	0
s_5	0	3	1
s_6	1	0	1

(a)



(b)

Perfect Phylogeny (Bounded Number of States), Figure 1
a An example of a character state matrix M for $S = \{s_1, s_2, \ldots, s_6\}$ and $C = \{c_1, c_2, c_3\}$ with $r_1 = 3$, $r_2 = 4$, and $r_3 = 2$, i. e., $r = 4$. b A perfect phylogeny for (S, C). For convenience, the states of all three characters for each object are shown

other taxa, populations, proteins, genes, natural languages, etc.) believed to have been produced by an evolutionary process. One of the most widely used techniques for reconstructing a phylogenetic tree is to represent the objects as vectors of character states and look for a tree that clusters objects which have a lot in common. The Perfect Phylogeny Problem can be regarded as the ideal special case of this approach in which the given data contains no errors, evolution is tree-like, and each character state can emerge only once in the evolutionary history.

However, data obtained experimentally seldom admits a perfect phylogeny, so various optimization versions of the problem such as *maximum parsimony* and *maximum compatibility* are often considered in practice; as might be expected, these strategies generally lead to NP-complete problems, but there exist many heuristics that work well for most inputs. See, e. g. [4,5,12], for a further discussion and references. Nevertheless, algorithms for The Perfect Phylogeny Problem may be useful even when the data does not admit a perfect phylogeny, for example if there exists a perfect phylogeny for $m - O(1)$ of the characters in C. In fact, in one crucial step of their proposed character-based methodology for determining the evolutionary history of a set of related natural languages, Warnow, Ringe, and Taylor [14] consider all subsets of C in decreasing order of cardinality, repeatedly applying the algorithm of [10] until a largest subset of C which admits a perfect phylogeny is found. The ideas behind the algorithms of [1] and [10] have also been utilized and extended by Fernández-Baca and Lagergren [6] in their algorithm for computing *near-perfect phylogenies* in which the constraints on the output have been relaxed in order to permit non-perfect phylogenies whose so-called penalty score is less than or equal to a prespecified parameter q (see [6] for details).

The motivation for considering a bounded number of states is that characters based on directly observable traits are, by the way they are defined, naturally bounded by some small number (often 2). When biomolecular data is used to define characters, the number of allowed states is typically bounded by a constant; e. g., $r = 2$ for SNP markers, $r = 4$ for DNA or RNA sequences, or $r = 20$ for amino-acid sequences (see also Encyclopedia ► Directed Perfect Phylogeny (Binary Characters)). Moreover, characters with $r = 2$ can be useful in comparative linguistics [8].

Open Problems

An important open problem is to determine whether the running time of the algorithm of Kannan and Warnow [10] can be improved. As pointed out in [5], it would be especially interesting to find out if The Perfect Phylogeny Problem is solvable in $O(2^{2r} nm)$ time for any r, or more generally, in $O(f(r) \cdot nm)$ time, where f is a function of r which does not depend on n or m, since this would match the fastest known algorithm for the special case $r = 2$ (see Table 1). Another open problem is to establish lower bounds on the computational complexity of The Perfect Phylogeny Problem with a bounded number of states.

Cross References

► Directed Perfect Phylogeny (Binary Characters)
► Perfect Phylogeny Haplotyping

Acknowledgments

Supported in part by Kyushu University, JSPS (Japan Society for the Promotion of Science), and INRIA Lille – Nord Europe.

Recommended Reading

1. Agarwala, R., Fernández-Baca, D.: A polynomial-time algorithm for the perfect phylogeny problem when the number of character states is fixed. SIAM J. Comput. **23**, 1216–1224 (1994)
2. Bodlaender, H.L., Fellows, M.R., Warnow, T.: Two strikes against perfect phylogeny. In: Proceedings of the 19th International Colloquium on Automata, Languages and Programming (ICALP 1992). Lecture Notes in Computer Science, vol. 623, pp. 273–283. Springer, Berlin (1992)
3. Dress, A., Steel, M.: Convex tree realizations of partitions. Appl. Math. Lett. **5**, 3–6 (1992)
4. Felsenstein, J.: Inferring Phylogenies. Sinauer Associates, Inc. Sunderland, Massachusetts (2004)
5. Fernández-Baca, D.: The Perfect Phylogeny Problem. In: Cheng, X., Du D.-Z. (eds.) Steiner Trees in Industry, pp. 203–234. Kluwer Academic Publishers, Dordrecht, Netherlands (2001)
6. Fernández-Baca, D., Lagergren, J.: A polynomial-time algorithm for near-perfect phylogeny. SIAM J. Comput. **32**, 1115–1127 (2003)
7. Gusfield, D.M.: Efficient algorithms for inferring evolutionary trees. Networks **21**, 19–28 (1991)
8. Kanj, I.A., Nakhleh, L., Xia, G.: Reconstructing evolution of natural languages: Complexity and parametrized algorithms. In: Proceedings of the 12th Annual International Computing and Combinatorics Conference (COCOON 2006). Lecture Notes in Computer Science, vol. 4112, pp. 299–308. Springer, Berlin (2006)
9. Kannan, S., Warnow, T.: Inferring evolutionary history from DNA sequences. SIAM J. Comput. **23**, 713–737 (1994)
10. Kannan, S., Warnow, T.: A fast algorithm for the computation and enumeration of perfect phylogenies. SIAM J. Comput. **26**, 1749–1763 (1997)
11. McMorris, F.R.: On the compatibility of binary qualitative taxonomic characters. Bull. Math. Biol. **39**, 133–138 (1977)
12. Setubal, J.C., Meidanis, J.: Introduction to Computational Molecular Biology. PWS Publishing Company, Boston (1997)
13. Steel, M.A.: The complexity of reconstructing trees from qualitative characters and subtrees. J. Classification **9**, 91–116 (1992)
14. Warnow, T., Ringe, D., Taylor, A.: Reconstructing the evolutionary history of natural languages. In: Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'96), pp. 314–322 (1996)

Perfect Phylogeny Haplotyping

2005; Ding, Filkov, Gusfield

GIUSEPPE LANCIA
Department of Mathematics and Computer Science, University of Udine, Udine, Italy

Keywords and Synonyms

Alleles phasing

Problem Definition

In the context of the *perfect phylogeny haplotyping* (PPH) problem, each vector $h \in \{0,1\}^m$ is called a *haplotype*, while each vector $g \in \{0,1,2\}^m$ is called a *genotype*. Haplotypes are binary encodings of DNA sequences, while genotypes are ternary encodings of *pairs* of DNA sequences (one sequence for each of the two homologous copies of a certain chromosome).

Two haplotypes h' and h'' are said to *resolve* a genotype g if, at each position j: (i) if $g_j \in \{0,1\}$ then both $h'_j = g_j$ and $h''_j = g_j$; (ii) if $g_j = 2$ then either $h'_j = 0$ and $h''_j = 1$ or $h'_j = 1$ and $h''_j = 0$. If h' and h'' resolve g, we write $g = h' + h''$. An instance of the PPH problem consists of a set $G = \{g^1, g^2, \dots, g^n\}$ of genotypes. A set H of haplotypes such that, for each $g \in G$, there are $h', h'' \in H$ with $g = h' + h''$, is called a *resolving set* for G.

A *perfect phylogeny* for a set H of haplotypes is a rooted tree T for which

- the set of leaves is H and the root is labeled by some binary vector r;
- each index $j \in \{1, \dots, m\}$ labels exactly one edge of T;
- if an edge e is labeled by an index k, then, for each leaf h that can be reached from the root via a path through e, it is $h_k \neq r_k$.

Without loss of generality, it can be assumed that the vector labeling the root is $r = 0$. Within the PPH problem, T is meant to represent the evolution of the sequences at the leaves from a common ancestral sequence (the root). Each edge labeled with an index represents a point in time when a mutation happened at a specific site. This model of evolution is also known as *coalescent* [11]. It can be shown that a perfect phylogeny for H exists if and only if for all choices of four haplotypes $h^1, \dots, h^4 \in H$ and two indices i, j,

$$\{h_i^a\, h_j^a\,, 1 \le a \le 4\} \neq \{00, 01, 10, 11\}\,.$$

Given the above definitions, the problem surveyed in this entry is the following:

Perfect Phylogeny Haplotyping Problem (PPH): Given a set G of genotypes, find a resolving set H of haplotypes and a perfect phylogeny T for H, or determine that such a resolving set does not exist.

In a slightly different version of the above problem, one may require to find all perfect phylogenies for H instead of just one (in fact, all known algorithms for PPH do find all perfect phylogenies).

The perfect phylogeny problem was introduced by Gusfield [7], who also proposed a nearly linear-time $O(nm\,\alpha(nm))$-algorithm for its solution (where $\alpha()$ is the extremely slowly growing inverse Ackerman function).

The algorithm resorted to a reduction to a complex procedure for the *graph realization* problem (Bixby and Wagner [2]), of very difficult understanding and implementation. Later approaches for PPH proposed much simpler, albeit slower, $O(nm^2)$-algorithms (Bafna et al. [1]; Eskin et al. [6]). However, a major question was left open: *does there exist a linear-time algorithm for PPH*? In [7], Gusfield conjectured that this should be the case. The 2005 algorithm by Ding, Filkov, and Gusfield [5] surveyed in this entry settles the above conjecture in the affirmative.

Key Results

The main idea of the algorithm is to find the maximal subgraphs that are common to all PPH solutions. Let us call *P-class* a maximal sub-graph of all PPH trees for G. The authors show that each P-class consists of two sub-trees which, in each PPH solution, can appear in either one of two possible ways (called *flips* of the P-class) with respect to any fixed P-class taken as a reference. Hence, if there are k P-classes, there are 2^{k-1} distinct PPH solutions.

The algorithm resorts to an original and effective data structure, called the *shadow tree*, which gives an implicit representation of all P-classes. The data structure is built incrementally, by processing one genotype at a time. The total cost for building and updating the shadow tree is linear in the input size (i. e., in nm). A detailed description of the shadow tree requires a rather large number of definitions, possibly accompanied by figures and examples. Here, we will introduce only its basic features, those that allow us to state the main theorems of [5].

The shadow tree is a particular type of directed rooted tree, which contains both *edges* and *links* (strictly speaking, the latter are just arcs, but they are called links to underline their specific use in the algorithm). The edges are of two types: *tree-edges* and *shadow-edges*, and are associated to the indices $\{1, \dots, m\}$. For each index i, there is a tree-edge labeled t_i and a shadow-edge labeled s_i. Both edges and links are oriented, with their head closer to the root than their tail. Other than the root, each node of the shadow tree is the endpoint of exactly one tree-edge or one shadow-edge (while the root is the head of two "dummy" links). The links are used to connect certain tree- and shadow-edges. A link can be either *free* or *fixed*. The head of a free link can still be changed during the execution of the algorithm, but once a link is fixed, it cannot be changed any more.

Tree-edges, shadow-edges and *fixed* links are organized into *classes*, which are sub-graphs of the shadow tree. Each fixed link is contained in exactly one class, while each free link connects one class to another, called its *parent*. For each index i, if the tree-edge t_i is in a class X, then the shadow-edge s_i is in X as well, so that a class can be seen as a pair of "twin" sub-trees of the shadow tree. The free links point out from the root of the sub-trees (the *class roots*). Classes change during the running of the algorithm. Specifically, classes are *created* (containing a single tree- and shadow-edge) when a new genotype is processed; a class can be *merged* with its parent, by fixing a pair of free edges; finally, a class can be *flipped*, by switching the heads of the two free links that connect the class roots to the parent class.

A tree T is said to be "*contained in*" a shadow tree if T can be obtained by flipping some classes in the shadow tree, followed by contracting all links and shadow-edges. Let us call *contraction* of a class the sub-graph (consisting of a pair of sub-trees, made of tree-edges only) that is obtained from a class X of the shadow tree by contracting all fixed links and shadow-edges of X. The following are the main results obtained in [5]:

Proposition 1 *Every P-class can be obtained by contraction of a class of the final shadow tree produced by the algorithm. Conversely, every contraction of a class of the final shadow tree is a P-class.*

Theorem 1 *Every PPH solution is contained in the final shadow tree produced by the algorithm. Conversely, every tree contained in the final shadow tree is a distinct PPH solution.*

Theorem 2 *The total time required for building and updating the shadow tree is* $O(nm)$.

Applications

The PPH problem arises in the context of *Single Nucleotide Polymorphisms* (SNP's) analysis in human genomes. A SNP is the site of a single nucleotide which varies in a statistically significant way in a population. The determination of SNP locations and of common SNP patterns (haplotypes) are of uttermost importance. In fact, SNP analysis is used to understand the nature of several genetic diseases, and the international Haplotype Map Project is devoted to SNP study (Helmuth [9]).

The values that a SNP can take are called its *alleles*. Almost all SNPs are bi-allelic, i. e., out of the four nucleotides A, C, T, G, only two are observed at any SNP. Humans are *diploid* organisms, with DNA organized in pairs of chromosomes (of paternal and of maternal origin). The sequence of alleles on a chromosome copy is called a *haplotype*. Since SNPs are bi-allelic, haplotypes can be encoded as binary strings. For a given SNP, an individual can be either *homozygous*, if both parents contributed the same allele, or *heterozygous*, if the paternal and maternal alleles are different.

Haplotyping an individual consists of determining his two haplotypes. Haplotyping a population consists of haplotyping each individual of the population. While it is today economically infeasible to determine the haplotypes directly, there is a cheap experiment which can determine the (less informative and often ambiguous) *genotypes*.

A genotype of an individual contains the *conflated* information about the two haplotypes. For each SNP, the genotype specifies which are the two (possibly identical) alleles, but does not specify their origin (paternal or maternal). The ternary encoding that is used to represent a genotype g has the following meaning: at each SNP j, it is $g_j = 0$ (respectively, 1) if the individual is homozygous for the allele 0 (respectively, 1), and $g_j = 2$ if the individual is heterozygous. There may be many possible pairs of haplotypes that justify a particular genotype (there are 2^{k-1} pairs of haplotypes that can resolve a genotype with k heterozygous SNPs). Given a set of genotypes, in order to determine the correct resolving set out of the exponentially many possibilities, one imposes some "biologically meaningful" constraints that the solution must possess. The perfect phylogeny model (coalescent) requires that the resolving set must fit a particular type of evolutionary tree. That is, all haplotypes should descend from some ancestral haplotype, via mutations that happened (only once) at specific sites over time. The coalescent model is accurate especially for short haplotypes (for longer haplotypes there is also another type of evolutionary event, *recombination*, that should be taken into account).

The linear-time PPH algorithm is of significant practical value in two respects. First, there are instances of the problem where the number of SNPs considered is fairly large (genotypes can extend over several kilo-bases). For these long instances, the advantage of an $O(nm)$ algorithm with respect to the previous $O(nm^2)$ approach is evident. On the other hand, when genotypes are relatively short, the benefit of using the linear-time algorithm is not immediately evident (both algorithms run extremely quickly). Nevertheless, there are situations in which one has to solve a large set of haplotyping problems, where each single problem is defined over short genotypes. For instance, this is the case in which one examines all "small" subsets of SNPs in order to determine the subsets for which there is a PPH solution. In this type of application, the gain of efficiency with the use of the linear-time PPH algorithm is significant (Chung and Gusfield [4]; Wiuf [14]).

Open Problems

A linear-time algorithm is the best possible for PPH, and no open problems are listed in [5].

Experimental Results

The algorithm has been implemented in C and its performance has been compared with the previous fastest PPH algorithm, i. e. DPPH (Bafna et al. [1]). In the case of $m = 2000$ and $n = 1000$, the algorithm is about 250-times faster than DPPH, and is capable of solving an instance in an average time of 2 seconds, versus almost 8 minutes needed by DPPH (on a "standard" 2005 Personal Computer). The smaller instances (e. g., with $m = 50$ SNPs) are such that the superior performance of the algorithm is not as evident, with an average running time of 0.07 seconds versus 0.2 seconds. However, as already remarked, when the small instances are executed within a loop, the speed-up turns out to be again of two or more orders of magnitude.

Data Sets

The data sets used in [5] have been generated by the program *ms* (Hudson [12]), which is the widely used standard for instance generation reflecting the coalescent model of SNP sequence evolution. Real-life instances can be found at the HapMap web site http://www.hapmap.org.

URL to Code

http://wwwcsif.cs.ucdavis.edu/~gusfield/lpph/

Cross References

► Directed Perfect Phylogeny (Binary Characters)
► Perfect Phylogeny (Bounded Number of States)

Recommended Reading

For surveys about computational haplotyping problems in general, see Bonizzoni et al. [3], Gusfield and Orzack [8], Halldorsson et al. [10], and Lancia [13].

1. Bafna, V., Gusfield, D., Lancia, G., Yooseph, S.: Haplotyping as perfect phylogeny: a direct approach. J. Comput. Biol. **10**(3–4), 323–340 (2003)
2. Bixby, R.E., Wagner, D.K.: An almost linear-time algorithm for graph realization. Math. Oper. Res. **13**, 99–123 (1988)
3. Bonizzoni, P., Della Vedova, G., Dondi, R., Li, J.: The haplotyping problem: an overview of computational models and solutions. J. Comput. Sci. Technol. **19**(1), 1–23 (2004)
4. Chung, R.H., Gusfield, D.: Empirical exploration of perfect phylogeny haplotyping and haplotypes. In: Proceedings of Annual International Conference on Computing and Combinatorics (COCOON). Lecture Notes in Computer Science, vol. 2697, pp. 5–9. Springer, Berlin (2003)
5. Ding, Z., Filkov, V., Gusfield, D.: A linear-time algorithm for the perfect phylogeny haplotyping problem. In: Proceedings of the Annual International Conference on Computational

Molecular Biology (RECOMB), New York, 2005. ACM Press, New York (2005)
6. Eskin, E., Halperin, E., Karp, R.: Efficient reconstruction of haplotype structure via perfect phylogeny. J. Bioinform. Comput. Biol. **1**(1), 1–20 (2003)
7. Gusfield, D.: Haplotyping as perfect phylogeny: Conceptual framework and efficient solutions. In: Myers, G., Hannenhalli, S., Istrail, S., Pevzner, P., Waterman, M. (eds.) Proceedings of the Annual International Conference on Computational Molecular Biology (RECOMB), New York, 2002, pp. 166–175. ACM Press (2002)
8. Gusfield, D. Orzack, S.H.: Haplotype inference. In: Aluru S. (ed) Handbook of Computational Molecular Biology, pp. 1–28. Champman and Hall/CRC-press, Boca Raton (2005)
9. Helmuth, L.: Genome research: Map of the human genome 3.0. Science **293**(5530), 583–585 (2001)
10. Halldorsson, B.V., Bafna, V., Edwards, N., Lippert, R., Yooseph, S., Istrail, S.: A survey of computational methods for determining haplotypes. In: Computational methods for SNP and haplotype inference: DIMACS/RECOMB satellite workshop. Lecture Notes in Computer Science, vol. 2983, pp. 26–47. Springer, Berlin (2004)
11. Hudson, R.: Gene genealogies and the coalescent process. Oxf. Surv. Evol. Biol. **7**, 1–44 (1990)
12. Hudson, R.: Generating samples under the wright-fisher neutral model of genetic variation. Bioinformatics **18**(2), 337–338 (2002)
13. Lancia, G.: The phasing of heterozygous traits: Algorithms and complexity. Comput. Math. Appl. **55**(5), 960–969 (2008)
14. Wiuf, C.: Inference on recombination and block structure using unphased data. Genetics **166**(1), 537–545 (2004)

Performance Analysis

► Distance-Based Phylogeny Reconstruction (Optimal Radius)

Performance-Driven Clustering

1993; Rajaraman, Wong

RAJMOHAN RAJARAMAN
Department of Computer Science,
Northeastern University,
Boston, MA, USA

Keywords and Synonyms

Circuit partitioning; Circuit clustering

Problem Definition

Circuit partitioning consists of dividing the circuit into parts each of which can be implemented as a separate component (e. g., a chip), that satisfies the design constraints. The work of Rajaraman and Wong [5] considers the problem of dividing a circuit into components, subject to area constraints, such that the maximum delay at the outputs is minimized.

A combinational circuit can be represented as a directed acyclic graph $G = (V, E)$, where V is the set of nodes, and E is the set of directed edges. Each node represents a gate in the network and each edge (u, v) in E represents an interconnection between gates u and v in the network. The *fanin* of a node is the number of edges incident into it, and the *fanout* of of a node is the number of edges incident out of it. A *primary input (PI)* is a node with fanin 0, while a *primary output (PO)* is a node with fanout 0. Each node has a *weight* and a *delay* associated with it.

Definition 1 A clustering of a network $G = (V, E)$ is a triple (H, ϕ, Σ), where
1. $H = (V', E')$ is a directed acyclic graph.
2. φ is a function mapping V' to V such that
   - For every edge $(u', v') \in E'$, $(\phi(u'), \phi(v')) \in E$.
   - For every node $v' \in V'$ and edge $(u, \phi(v')) \in E$, there exists a unique $u' \in V'$ such that $\phi(u') = u$ and $(u', v') \in E'$.
   - For every PO node $v \in V$, there exists a unique $v' \in V'$ such that $\phi(v') = v$.
3. Σ is a partition of V'.

Let $\Gamma = (H = (V', E'), \phi, \Sigma)$ be a clustering of G. For $v \in V$, $v' \in V'$, if $\phi(v') = v$, we call v' a *copy* of v. The set V' consists of all the copies of the nodes in V that appear in the clustering. A node v' is a PI (respectively, PO) in Γ if $\phi(v')$ is a PI (respectively, PO) in G. It follows from the definition of φ that H is logically equivalent to G.

The weights and delays on the individual nodes in G yield weights and delays of nodes in H' and a delay for the clustering Γ. The weight (respectively, delay) of an node v' in V' is the weight (respectively, delay) of $\phi(v)$. The weight of any cluster $C \in \Sigma$, denoted by $W(C)$, is the sum of the weights of the nodes in C. The delay of a clustering is given by the general delay model of Murgai et al. [3], which is as follows. The delay of an edge $(u', v') \in E'$ is D (which is a given parameter) if u' and v' belong to different elements of Σ and zero otherwise. The delay along a path in H' is simply the sum of the delays of the edges of the path. Finally, the delay of Γ is the delay of a maximum-delay path in H', among all the paths from a PI node to a PO node in H'.

Definition 2 Given a combinational network $G = (V, E)$ with weight function $w \colon V \to R^+$, weight capacity M and a delay function $\delta \colon V \to R^+$, we say that a clustering $\Gamma = (H, \phi, \Sigma)$ is *feasible* if for every cluster $C \in \Sigma$, $W(C)$

is at most M. The *circuit clustering problem* is to compute a feasible clustering Γ of G such that the delay of Γ is minimum among all feasible clusterings of G.

An early work of Lawler et al. [2] presented a polynomial-time optimal algorithm for the circuit clustering problem in the special case where all the gate delays are zero (i. e., $\delta(v) = 0$ for all v).

Key Results

Rajaraman and Wong [5] presented an optimal polynomial-time algorithm for the circuit clustering problem under the general delay model.

Theorem 1 *There exists an algorithm that computes an optimal clustering for the circuit clustering problem in $O(n^2 \log n + nm)$ time, where n and m are the vertices and edges, respectively, of the given combinational network.*

This result can be extended to compute optimal clusterings under any monotone clustering constraint. A clustering constraint is monotone if any connected subset of nodes in a feasible cluster is also monotone [2].

Theorem 2 *The circuit clustering problem can be solved optimally under any monotone clustering constraint in time polynomial in the size of the circuit.*

Applications

Circuit partitioning/clustering is an important component of very large scale integration design. One application of the circuit clustering problem formulated above is to implement a circuit on multiple field programmable gate array chips. The work of Rajaraman and Wong focused on clustering combinational circuits to minimize delay under area constraints. Related studies have considered other important constraints, such as pin constraints [1] and a combination of area and pin constraints [6]. Further work has also included clustering sequential circuits (as opposed to combinational circuits) with the objective of minimizing the clock period [4].

Experimental Results

Rajaraman and Wong reported experimental results on five ISCAS (International Symposium on Circuits and Systems) circuits. The number of nodes in these circuits ranged from 196 to 913. They reported the maximum delay of the clusterings and running times of their algorithm on a Sun Sparc workstation.

Cross References

▶ FPGA Technology Mapping

Recommended Reading

1. Cong, J., Ding, Y.: An optimal technology mapping algorithm for delay optimization in lookup-table based fpga design. In: Proceedings of IEEE International Conference on Computer-Aided Design, 1992, pp. 48–53
2. Lawler, E.L., Levitt, K.N., Turner, J.: Module clustering to minimize delay in digital networks. IEEE Trans. Comput. **C-18**, 47–57 (1966)
3. Murgai, R., Brayton, R.K., Sangiovanni-Vincentelli, A.: On clustering for minimum delay/area. In: Proceedings of IEEE International Conference on Computer-Aided Design, 1991, pp. 6–9
4. Pan, P., Karandikar, A.K., Liu, C.L.: Optimal clock period clustering for sequential circuits with retiming. IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. **17**, 489–498 (1998)
5. Rajaraman, R., Wong, D.F.: Optimum clustering for delay minimization. IEEE Trans. Comput.-Aided Des. Integr. Circ. Syst. **14**, 1490–1495 (1995)
6. Yang, H.H., Wong, D.F.: Circuit clustering for delay minimization under area and pinconstraints. IEEE Trans. Comput.-Aided Des. Integr. Circ. Syst. **16**, 976–986 (1997)

Phylogenetic Tree Construction from a Distance Matrix

1989; Hein

JESPER JANSSON
Ochanomizu University, Tokyo, Japan

Keywords and Synonyms

Phylogenetic tree construction from a dissimilarity matrix

Problem Definition

Let n be a positive integer. A *distance matrix of order n* (also called a *dissimilarity matrix of order n*) is a matrix D of size $(n \times n)$ which satisfies: (1) $D_{i,j} > 0$ for all $i, j \in \{1, 2, \ldots, n\}$ with $i \neq j$; (2) $D_{i,j} = 0$ for all $i, j \in \{1, 2, \ldots, n\}$ with $i = j$; and (3) $D_{i,j} = D_{j,i}$ for all $i, j \in \{1, 2, \ldots, n\}$.

Below, all trees are assumed to be unrooted and edge-weighted. For any tree $\mathcal{T}$, the *distance* between two nodes u and v in $\mathcal{T}$ is defined as the sum of the weights of all edges on the unique path in $\mathcal{T}$ between u and v, and is denoted by $d^{\mathcal{T}}_{u,v}$. A tree $\mathcal{T}$ is said to *realize* a given distance matrix D of order n if and only if it holds that $\{1, 2, \ldots, n\}$ is a subset of the nodes of $\mathcal{T}$ and $d^{\mathcal{T}}_{i,j} = D_{i,j}$ for all $i, j \in \{1, 2, \ldots, n\}$. Finally, a distance matrix D is called *additive* or *tree-realizable* if and only if there exists a tree which realizes D.

Problem 1 (The Phylogenetic Tree from Distance Matrix Problem)
INPUT: *An distance matrix D of order n.*
OUTPUT: *A tree which realizes D and has the smallest possible number of nodes, if D is additive; otherwise, null.*

See Fig. 1 for an example.

In the time complexities listed below, the time needed to input all of D is not included. Instead, $O(1)$ is charged to the running time whenever an algorithm requests to know the value of any specified entry of D.

Key Results

Several authors have independently shown how to solve The Phylogenetic Tree from Distance Matrix Problem. The fastest of these algorithms run in $O(n^2)$ time[1]:

Theorem 1 ([2,4,5,7,15]) *There exists an algorithm which solves The Phylogenetic Tree from Distance Matrix Problem in $O(n^2)$ time.*

Although the algorithms are different, it can be proved that:

Theorem 2 ([8,15]) *For any given distance matrix, the solution to The Phylogenetic Tree from Distance Matrix Problem is unique.*

Furthermore, the algorithms referred to in Theorem 1 have optimal running time since any algorithm for The Phylogenetic Tree from Distance Matrix Problem must in the worst case query all $\Omega(n^2)$ entries of D to make sure that D is additive. However, if it is known in advance that the input distance matrix is additive then the time complexity improves, as shown by Hein [9]:

Theorem 3 ([9,12]) *There exists an algorithm which solves The Phylogenetic Tree from Distance Matrix Problem restricted to additive distance matrices in $O(kn \log_k n)$ time, where k is the maximum degree of the tree that realizes the input distance matrix*[2].

The algorithm of Hein [9] starts with a tree containing just two nodes and then successively inserts each node i into the tree by repeatedly choosing a pair of existing nodes and computing where on the path between them that i should be attached, until i's position has been determined. (The same basic technique is used in the $O(n^2)$-time algorithm of Waterman et al. [15] referenced to by Theorem 1 above, but the algorithm of Hein selects paths which are more efficient at discriminating between the possible positions for i.)

The lower bound corresponding to Theorem 3 is given by:

Theorem 4 ([10]) *The Phylogenetic Tree from Distance Matrix Problem restricted to additive distance matrices requires $\Omega(kn \log_k n)$ queries to the distance matrix D, where k is the maximum degree of the tree that realizes D, even if restricted to trees in which all edge weights are equal to 1.*

Finally, note that the following special case is easily solvable in linear time:

Theorem 5 ([5]) *There exists an $O(n)$-time algorithm which solves The Phylogenetic Tree from Distance Matrix Problem restricted to additive distance matrices for which the realizing tree contains two leaves only and has all edge weights equal to 1.*

Applications

The main application of The Phylogenetic Tree from Distance Matrix Problem is in the construction of a tree (a so-called *phylogenetic tree*) that represents evolutionary relationships among a set of studied objects (e. g., species or other taxa, populations, proteins, genes, etc.). Here, it is assumed that the objects are indeed related according to a tree-like branching pattern caused by an evolutionary process and that their true pairwise evolutionary distances are proportional to the measured pairwise dissimilarities. See, e. g., [1,6,7,14,15] for examples and many references as well as discussions on how to estimate pairwise dissimilarities based on biological data. Other applications of The Phylogenetic Tree from Distance Matrix Problem can be found in psychology, for example to describe semantic memory organization [1], in comparative linguistics to infer the evolutionary history of a set of languages [11], or in the study of the filiation of manuscripts to trace how manuscript copies of a text (whose original version may have been lost) have evolved in order to identify discrepancies among them or to reconstruct the original text [1,3,13].

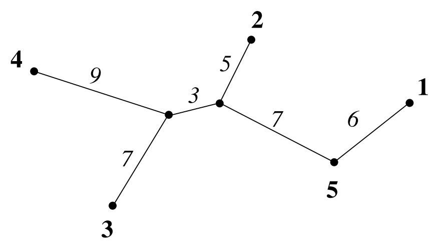
In general, real data seldom forms additive distance matrices [15]. Therefore, in practice, researchers consider optimization versions of The Phylogenetic Tree from Distance Matrix Problem which look for a tree that "almost" realizes D. Many alternative definitions of "almost" have been proposed, and numerous heuristics and approxima-

[1] See [5] for a short survey of older algorithms which do not run in $O(n^2)$ time.

[2] For this case, the Culberson-Rudnicki algorithm [5] runs in $O(n^{3/2}\sqrt{k})$ time for trees in which all edge weights are equal to 1, and not in $O(kn \log_k n)$ time as claimed in [5]. See [12] for a counterexample to [5] and a correct analysis.

$$D = \begin{bmatrix} 0 & 18 & 23 & 25 & 6 \\ 18 & 0 & 15 & 17 & 12 \\ 23 & 15 & 0 & 16 & 17 \\ 25 & 17 & 16 & 0 & 19 \\ 6 & 12 & 17 & 19 & 0 \end{bmatrix}$$

a



b

Phylogenetic Tree Construction from a Distance Matrix, Figure 1
a An additive distance matrix *D* of order 5. b A tree $\mathcal{T}$ which realizes *D*. Here, {1, 2, . . . , 5} forms a subset of the nodes of $\mathcal{T}$

tion algorithms have been developed. A comprehensive description of some of the most popular distance-based methods for phylogenetic reconstruction as well as more background information can be found in, e. g., Chapt. 11 of [6] or Chapt. 4 of [14]. See also [1] and [16] and the references therein.

Cross References

► Distance-Based Phylogeny Reconstruction (Fast-Converging)
► Distance-Based Phylogeny Reconstruction (Optimal Radius)

Acknowledgments

Supported in part by Kyushu University, JSPS (Japan Society for the Promotion of Science), and INRIA Lille – Nord Europe.

Recommended Reading

1. Abdi, H.: Additive-tree representations. In: Dress, A., von Haeseler, A. (eds.) Trees and Hierarchical Structures: Proceedings of a conference held at Bielefeld, FRG, Oct. 5–9th, 1987. Lecture Notes in Biomathematics, vol. 84, pp. 43–59. Springer (1990)
2. Batagelj, V., Pisanski, T., Simões-Pereira, J.M.S.: An algorithm for tree-realizability of distance matrices. Int. J. Comput. Math. **34**, 171–176 (1990)
3. Bennett, C.H., Li, M., Ma, B.: Chain letters and evolutionary histories. Sci. Am. **288**, 76–81 (2003)
4. Boesch, F.T.: Properties of the distance matrix of a tree. Quarterly Appl. Math. **26**, 607–609 (1968)
5. Culberson, J.C., Rudnicki, P.: A fast algorithm for constructing trees from distance matrices. Inf. Process. Lett. **30**, 215–220 (1989)
6. Felsenstein, J.: Inferring Phylogenies. Sinauer Associates, Inc. (2004)
7. Gusfield, D.M.: Algorithms on Strings, Trees, and Sequences. Cambridge University Press, New York (1997)
8. Hakimi, S.L., Yau, S.S.: Distance matrix of a graph and its realizability. Quarterly Appl. Math. **22**, 305–317 (1964)
9. Hein, J.: An optimal algorithm to reconstruct trees from additive distance data. Bull. Math. Biol. **51**, 597–603 (1989)
10. King, V., Zhang, L., Zhou, Y.: On the complexity of distance-based evolutionary tree construction. In: Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2003), pp. 444–453 (2003)
11. Nakhleh, L., Warnow, T., Ringe, D., Evans, S.N.: A comparison of phylogenetic reconstruction methods on an Indo-European dataset. Trans. Philol. Soc. **103**, 171–192 (2005)
12. Reyzin, L., Srivastava, N.: On the longest path algorithm for reconstructing trees from distance matrices. Inf. Process. Lett. **101**, 98–100 (2007)
13. The Canterbury Tales Project: University of Birmingham, Brigham Young University, University of Münster, New York University, Virginia Tech, and Keio University. Website: http://www.canterburytalesproject.org/
14. Warnow, T.: Some combinatorial optimization problems in phylogenetics. In: Lovász, L., Gyárfás, G., Katona, G., Recski, A., Székely, L. (eds.) Graph Theory and Combinatorial Biology. Bolyai Society Mathematical Studies, vol. 7, pp. 363–413. Bolyai János Matematikai Társulat (1999)
15. Waterman, M.S., Smith, T.F., Singh, M., Beyer, W.A.: Additive evolutionary trees. J. Theor. Biol. **64**, 199–213 (1977)
16. Wu, B.Y., K.-Chao, M., Tang, C.Y.: Approximation and exact algorithms for constructing minimum ultrametric trees from distance matrices. J. Combin. Optim. **3**, 199–211 (1999)

Phylogeny Reconstruction

► Distance-Based Phylogeny Reconstruction (Optimal Radius)

Planar Geometric Spanners

2005; Bose, Smid, Gudmundsson

JOACHIM GUDMUNDSSON[1], GIRI NARASIMHAN[2], MICHIEL SMID[3]
[1] DMiST, National ICT Australia Ltd, Alexandria, Australia
[2] Department of Computer Science, Florida International University, Miami, FL, USA
[3] School of Computer Science, Carleton University, Ottawa, ON, Canada

Keywords and Synonyms

Geometric network; Dilation; Detour

Problem Definition

Let S be a set of n points in the plane and let G be an undirected graph with vertex set S, in which each edge (u, v) has a weight, which is equal to the Euclidean distance $|uv|$ between the points u and v. For any two points p and q in S, their shortest-path distance in G is denoted by $\delta_G(p, q)$. If $t \geq 1$ is a real number, then G is a *t-spanner* for S if $\delta_G(p, q) \leq t|pq|$ for any two points p and q in S. Thus, if t is close to 1, then the graph G contains close approximations to the $\binom{n}{2}$ Euclidean distances determined by the pairs of points in S. If, additionally, G consists of $O(n)$ edges, then this graph can be considered a sparse approximation to the complete graph on S. The smallest value of t for which G is a t-spanner is called the *stretch factor* (or *dilation*) of G. For a comprehensive overview of geometric spanners, see the book by Narasimhan and Smid [16].

Assume that each edge (u, v) of G is embedded as the straight-line segment between the points u and v. The graph G is said to be *plane* if its edges intersect only at their common vertices.

In this entry, the following two problems are considered:

Problem 1 *Determine the smallest real number $t > 1$ for which the following is true: For every set S of n points in the plane, there exists a plane graph with vertex set S, which is a t-spanner for S. Moreover, design an efficient algorithm that constructs such a plane t-spanner.*

Problem 2 *Determine the smallest positive integer D for which the following is true: There exists a constant t, such that for every set S of n points in the plane, there exists a plane graph with vertex set S and maximum degree at most D, which is a t-spanner for S. Moreover, design an efficient algorithm that constructs such a plane t-spanner.*

Key Results

Let S be a finite set of points in the plane that is in *general position*, i. e., no three points of S are on a line and no four points of S are on a circle. The *Delaunay triangulation* of S is the plane graph with vertex set S, in which (u, v) is an edge if and only if there exists a circle through u and v that does not contain any point of S in its interior. (Since S is in general position, this graph is a triangulation.) The Delaunay triangulation of a set of n points in the plane can be constructed in $O(n \log n)$ time. Dobkin, Friedman and Supowit [10] were the first to show that the stretch factor of the Delaunay triangulation is bounded by a constant: They proved that the Delaunay triangulation is a t-spanner for $t = \pi(1 + \sqrt{5})/2$. The currently best known upper bound on the stretch factor of this graph is due to Keil and Gutwin [12]:

Theorem 1 *Let S be a finite set of points in the plane. The Delaunay triangulation of S is a t-spanner for S, for $t = 4\pi\sqrt{3}/9$.*

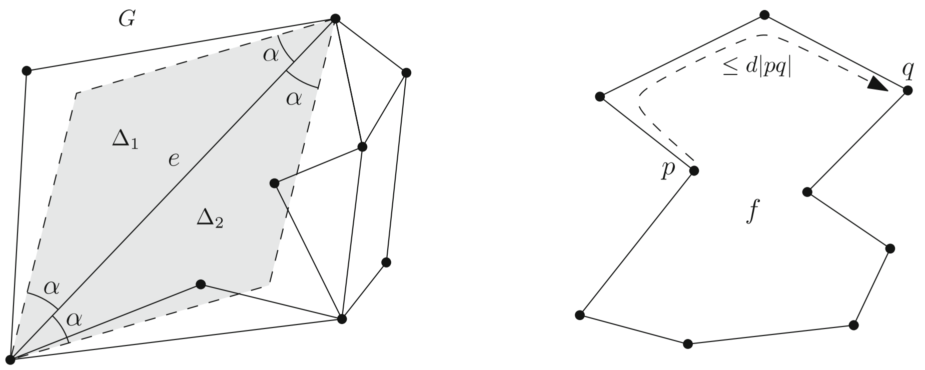
A slightly stronger result was proved by Bose et al. [3]. They proved that for any two points p and q in S, the Delaunay triangulation contains a path between p and q, whose length is at most $(4\pi\sqrt{3}/9)|pq|$ and all edges on this path have length at most $|pq|$.

Levcopoulos and Lingas [14] generalized the result of Theorem 1: Assume that the Delaunay triangulation of the set S is given. Then, for any real number $r > 0$, a plane graph G with vertex set S can be constructed in $O(n)$ time, such that G is a t-spanner for S, where $t = (1+1/r)4\pi\sqrt{3}/9$, and the total length of all edges in G is at most $2r + 1$ times the weight of a minimum spanning tree of S.

The Delaunay triangulation can alternatively be defined to be the dual of the *Voronoi diagram* of the set S. By considering the Voronoi diagram for a metric other than the Euclidean metric, a corresponding Delaunay triangulation is obtained. Chew [7] has shown that the Delaunay triangulation based on the Manhattan-metric is a $\sqrt{10}$-spanner (in this spanner, path-lengths are measured in the Euclidean metric). The currently best result for Problem 1 is due to Chew [8]:

Theorem 2 *Let S be a finite set of points in the plane, and consider the Delaunay triangulation of S that is based on the convex distance function defined by an equilateral triangle. This plane graph is a 2-spanner for S (where path-lengths are measured in the Euclidean metric).*

Das and Joseph [9] have generalized the result of Theorem 1 in the following way (refer to Fig. 1). Let G be a plane graph with vertex set S and let α be a real number with $0 < \alpha < \pi/2$. For any edge e of G, let Δ_1 and Δ_2 be the two isosceles triangles with base e and base angle α. The edge e is said to satisfy the *α-diamond property*, if at least one of the triangles Δ_1 and Δ_2 does not contain any point of S in its interior. The plane graph G is said to satisfy the *α-diamond property*, if every edge e of G satisfies this property. For a real number $d \geq 1$, G satisfies the *d-good polygon property*, if for every face f of G, and for every two vertices p and q on the boundary of f, such that the line segment joining them is completely inside f, the shortest path between p and q along the boundary of f has length at most $d|pq|$. Das and Joseph [9] proved that any plane



Planar Geometric Spanners, Figure 1
On the left, the α-diamond property is illustrated. At least one of the triangles Δ_1 and Δ_2 does not contain any point of *S* in its interior. On the right, the *d*-good polygon property is illustrated. *p* and *q* are two vertices on the same face *f* which can see each other. At least one of the two paths between *p* and *q* along the boundary of *f* has length at most $d|pq|$

graph satisfying both the α-diamond property and the d-good polygon property is a t-spanner, for some real number t that depends only on α and d. A slight improvement on the value of t was obtained by Lee [13]:

Theorem 3 *Let $\alpha \in (0, \pi/2)$ and $d \geq 1$ be real numbers, and let G be a plane graph that satisfies the α-diamond property and the d-good polygon property. Then, G is a t-spanner for the vertex set of G, where*

$$t = \frac{8(\pi - \alpha)^2 d}{\alpha^2 \sin^2(\alpha/4)} .$$

To give some examples, it is not difficult to show that the Delaunay triangulation satisfies the α-diamond property with $\alpha = \pi/4$. Drysdale et al. [11] have shown that the minimum weight triangulation satisfies the α-diamond property with $\alpha = \pi/4.6$. Finally, Lee [13] has shown that the greedy triangulation satisfies the α-diamond property with $\alpha = \pi/6$. Of course, any triangulation satisfies the d-good polygon property with $d = 1$.

Now consider Problem 2, that is, the problem of constructing plane spanners whose maximum degree is small. The first result for this problem is due to Bose et al. [2]. They proved that the Delaunay triangulation of any finite point set contains a subgraph of maximum degree at most 27, which is a t-spanner (for some constant t). Li and Wang [15] improved this result, by showing that the Delaunay triangulation contains a t-spanner of maximum degree at most 23. Given the Delaunay triangulation, the subgraphs in [2,15] can be constructed in $O(n)$ time. The currently best result for Problem 2 is by Bose et al. [6]:

Theorem 4 *Let S be a set of n points in the plane. The Delaunay triangulation of S contains a subgraph of maximum degree at most 17, which is a t-spanner for S, for some constant t. Given the Delaunay triangulation of S, this subgraph can be constructed in O(n) time.*

In fact, the result in [6] is more general:

Theorem 5 *Let S be a set of n points in the plane, let $\alpha \in (0, \pi/2)$ be a real number, and let G be a triangulation of S that satisfies the α-diamond property. Then, G contains a subgraph of maximum degree at most $14 + \lceil 2\pi/\alpha \rceil$, which is a t-spanner for S, where t depends only on α. Given the triangulation G, this subgraph can be constructed in O(n) time.*

Applications

Plane spanners have applications in on-line path-finding and routing problems that arise in, for example, geographic information systems and communication networks. In these application areas, the complete environment is not known, and routing has to be done based only on the source, the destination, and the neighborhood of the current position. Bose and Morin [4,5] have shown that, in this model, good routing strategies exist for plane graphs, such as the Delaunay triangulation and graphs that satisfy both the α-diamond property and the d-good polygon property. These strategies are competitive, in the sense that the paths computed have lengths that are within a constant factor of the Euclidean distance between the source and destination. Moreover, these routing strategies use only a limited amount of memory.

Open Problems

None of the results for Problems 1 and 2 that are mentioned in Sect. "Key Results" seem to be optimal. The following problems are open:

1. Determine the smallest real number t, such that the Delaunay triangulation of any finite set of points in the plane is a t-spanner. It is widely believed that $t = \pi/2$. By Theorem 1, $t \leq 4\pi\sqrt{3}/9$.
2. Determine the smallest real number t, such that a plane t-spanner exists for any finite set of points in the plane. By Theorem 2, $t \leq 2$. By taking S to be the set of four vertices of a square, it follows that t must be at least $\sqrt{2}$.
3. Determine the smallest integer D, such that the Delaunay triangulation of any finite set of points in the plane contains a t-spanner (for some constant t) of maximum degree at most D. By Theorem 4, $D \leq 17$. It follows from results in Aronov et al. [1] that the value of D must be at least 3.
4. Determine the smallest integer D, such that a plane t-spanner (for some constant t) of maximum degree at most D exists for any finite set of points in the plane. By Theorem 4 and results in [1], $3 \leq D \leq 17$.

Cross References

► Applications of Geometric Spanner Networks
► Dilation of Geometric Networks
► Geometric Spanners
► Sparse Graph Spanners

Recommended Reading

1. Aronov, B., de Berg, M., Cheong, O., Gudmundsson, J., Haverkort, H., Vigneron, A.: Sparse geometric graphs with small dilation. In: Proceedings of the 16th International Symposium on Algorithms and Computation. Lecture Notes in Computer Science, vol. 3827, pp. 50–59. Springer, Berlin (2005)
2. Bose, P., Gudmundsson, J., Smid, M.: Constructing plane spanners of bounded degree and low weight. Algorithmica **42**, 249–264 (2005)
3. Bose, P., Maheshwari, A., Narasimhan, G., Smid, M., Zeh, N.: Approximating geometric bottleneck shortest paths. Comput. Geom.: Theory Appl. **29**, 233–249 (2004)
4. Bose, P., Morin, P.: Competitive online routing in geometric graphs. Theor. Comput. Sci. **324**, 273–288 (2004)
5. Bose, P., Morin, P.: Online routing in triangulations. SIAM J. Comput. **33**, 937–951 (2004)
6. Bose, P., Smid, M., Xu, D.: Diamond triangulations contain spanners of bounded degree. In: Proceedings of the 17th International Symposium on Algorithms and Computation. Lecture Notes in Computer Science, vol. 4288, pp. 173–182. Springer, Berlin (2006)
7. Chew, L.P.: There is a planar graph almost as good as the complete graph. In: Proceedings of the 2nd ACM Symposium on Computational Geometry, pp. 169–177 (1986)
8. Chew, L.P.: There are planar graphs almost as good as the complete graph. J. Comput. Syst. Sci. **39**, 205–219 (1989)
9. Das, G., Joseph, D.: Which triangulations approximate the complete graph? In: Proceedings of the International Symposium on Optimal Algorithms. Lecture Notes in Computer Science, vol. 401, pp. 168–192. Springer, Berlin (1989)
10. Dobkin, D.P., Friedman, S.J., Supowit, K.J.: Delaunay graphs are almost as good as complete graphs. Discret. Comput. Geom. **5**, 399–407 (1990)
11. Drysdale, R.L., McElfresh, S., Snoeyink, J.S.: On exclusion regions for optimal triangulations. Discrete Appl. Math. **109**, 49–65 (2001)
12. Keil, J.M., Gutwin, C.A.: Classes of graphs which approximate the complete Euclidean graph. Discrete Comput. Geom. **7**, 13–28 (1992)
13. Lee, A.W.: Diamonds are a plane graph's best friend. Master's thesis, School of Computer Science, Carleton University, Ottawa (2004)
14. Levcopoulos, C., Lingas, A.: There are planar graphs almost as good as the complete graphs and almost as cheap as minimum spanning trees. Algorithmica **8**, 251–256 (1992)
15. Li, X.-Y., Wang, Y.: Efficient construction of low weighted bounded degree planar spanner. Int. J. Comput. Geom. Appl. **14**, 69–84 (2004)
16. Narasimhan, G., Smid, M.: Geometric Spanner Networks. Cambridge University Press, Cambridge, UK (2007)

Planarity Testing

1976; Booth, Lueker

GLENCORA BORRADAILE
Department of Computer Science, Brown University, Providence, RI, USA

Keywords and Synonyms

Planarity testing; Planar embedding

Problem Definition

The problem is to determine whether or not the input graph G is planar. The definition pertinent to planarity-testing algorithms is: G is planar if there is an *embedding* of G into the plane (vertices of G are mapped to distinct points and edges of G are mapped to curves between their respective endpoints) such that edges do not cross. Algorithms that test the planarity of a graph can be modified to obtain such an embedding of the graph.

Key Results

Theorem 1 *There is an algorithm that given a graph G with n vertices, determines whether or not G is planar in $O(n)$ time.*

The first linear-time algorithm was obtained by Hopcroft and Tarjan [5] by analyzing an iterative version of a recursive algorithm suggested by Auslander and Parter [1] and corrected by Goldstein [4]. The algorithm is based on the observation that a connected graph is planar if and only if all its biconnected components are planar. The recursive algorithm works with each biconnected component in turn: find a separating cycle C and partition the edges of G not in C; define a component of the partition as consisting of edges connected by a path in G that does not use an edge of C; and, recursively consider each cyclic component of the partition. If each component of the partition is planar and the components can be combined with C to give a planar graph, then G is planar.

Another method for determining planarity was suggested by Lempel, Even, and Cederbaum [6]. The algorithm starts with embedding a single vertex and the edges adjacent to this vertex. It then considers a vertex adjacent to one of these edges. For correctness, the vertices must be considered in a particular order. This algorithm was first implemented in $O(n)$ time by Booth and Lueker [2] using an efficient implementation of the PQ-trees data structure. Simpler implementations of this algorithm have been given by Boyer and Myrvold [3] and Shih and Hsu [8].

Tutte gave an algebraic method for giving a *straight-line embedding* of a graph that, if the input graph is 3-connected and planar, is guaranteed to generate a planar embedding. The key idea is to fix the vertices of one face of the graph to be the corners of a convex polygon and then embed every other vertex as the geometric average of its neighbors.

Applications

Planarity testing has applications to computer-aided circuit design and VLSI layout by determining whether a given network can be realized in the plane.

URL to Code

LEDA has an efficient implementation of the Hopcroft and Tarjan planarity testing algorithm [7]: http://www.algorithmic-solutions.info/leda_guide/graph_algorithms/planar_kuratowski.html

Cross References

▶ Fully Dynamic Planarity Testing

Recommended Reading

1. Auslander, L., Parter, S.V.: On imbedding graphs in the plane. J. Math. and Mech. **10**, pp. 517–523 (1961)
2. Booth, K.S., Lueker, G.S.: Testing for the consecutive ones property, interval graphs, and graph planarity using *PQ*-tree algorithms. J. Comp. Syst. Sci. **13**, pp. 335–379 (1976)
3. Boyer, J., Myrvold, W.: Stop minding your P's and Q's: A simplified O(n) planar embedding algorithm. In: SODA '99: Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms. Philadelphia, PA, USA, Society for Industrial and Applied Mathematics, pp. 140–146 (1999)
4. Goldstein, A.J.: An efficient and constructive algorithm for testing whether a graph can be embedded in the plane. In: Graph and Combinatorics Conf. (1963)
5. Hopcroft, J., Tarjan, R.: Efficient planarity testing. J. ACM **21**, pp. 549–568 (1974)
6. Lempel, A., Even, S., Cederbaum, I.: An algorithm for planarity testing of graphs. In: Rosentiehl, P. (ed.) Theory of Graphs: International Symposium. New York, Gordon and Breach, pp. 215–232 (1967)
7. Mehlhorn, K., Mutzel, P., Näher, S.: An implementation of the hopcroft and tarjan planarity test. Tech. Rep. MPI-I-93-151, Saarbrücken (1993)
8. Shih, W.-K., Hsu, W.-L.: A new planarity test. Theor. Comput. Sci. **223**, pp. 179–191 (1999)

Point Pattern Matching

2003; Ukkonen, Lemström, Mäkinen

VELI MÄKINEN, ESKO UKKONEN
Department of Computer Science, University of Helsinki, Helsinki, Finland

Keywords and Synonyms

Point set matching; Geometric matching; Geometric alignment; Largest common point set

Problem Definition

Let $\mathbb{R}$ denote the set of reals and $\mathbb{R}^d$ the d-dimensional real space. A finite subset of $\mathbb{R}^d$ is called a *point set*. The set of all point sets (subsets of $\mathbb{R}^d$) is denoted $\mathcal{P}(\mathbb{R}^d)$.

Point pattern matching problems ask for finding similarities between point sets under some transformations. In the basic set–up a *target* point set $T \subset \mathbb{R}^d$ and a *pattern* point set (*point pattern*) $P \subset \mathbb{R}^d$ are given, and the problem is to locate a subset I of T (if it exists) such that P *matches* I. Matching here means that P becomes exactly or approximately equal to I when a *transformation* from a given set $\mathcal{F}$ of transformations is applied on P.

Set $\mathcal{F}$ can be, for example, the set of all *translations* (a constant vector added to each point in P), or all compositions of translations and *rotations* (after a translation, each point is rotated with respect to a common origin; this preserves the distances and is also called a *rigid movement*), or all compositions of translations, rotations, and *scales* (after translating and rotating, distances to the common origin are multiplied by a constant).

The problem variant with exact matching, called the *Exact Point Pattern Matching (EPPM)* problem, requires that $f(P) = I$ for some $f \in \mathcal{F}$. In other words, the EPPM problem is to decide whether or not there is an allowed transformation f such that $f(P) \subset T$. For example, if $\mathcal{F}$

is the set of translations, the problem is simply to decide whether $P + t \subset T$ for some $t \in \mathbb{R}^d$.

Approximate matching is a better model of many situations that arise in practice. Then the quality of the matching between $f(P)$ and I is controlled using a *threshold parameter* $\varepsilon \geq 0$ and a *distance function* $\delta \colon (\mathcal{P}(\mathbb{R}^d), \mathcal{P}(\mathbb{R}^d)) \to \mathbb{R}$ for measuring distances between point sets. Given $\varepsilon \geq 0$, the *Approximate Point Pattern Matching (APPM)* problem is to determine whether there is a subset $I \subset T$ and a transformation $f \in \mathcal{F}$ such that $\delta(f(P), I) \leq \varepsilon$.

The choice of the distance function δ is another source of diversity in the problem statement. A variant requires that there is a *one-to-one* mapping between $f(P)$ and I, and each point p of $f(P)$ is ε-*close* to its one-to-one counterpart p^* in I, that is, $|p - p^*| \leq \varepsilon$. A commonly studied relaxed version uses matching under a *many-to-one* mapping: it is only required that each point of $f(P)$ has *some* point of I that is ε-*close*; this distance is also known as the *directed Hausdorff distance*. Still more variants come from the choice of the *norm* $|\cdot|$ to measure the distance between points.

Another form of approximation is obtained by allowing a minimum amount of unmatched points in P: The *Largest Common Point Set (LCP)* problem asks for the largest $I \subset T$ such that $I \subset f(P)$ for some $f \in \mathcal{F}$. In the *Largest Approximately Common Point Set (LACP)* problem each point $p^* \in I$ must occur ε-*close* to a point $p \in f(P)$.

Finally, a problem closely related to point pattern matching is to evaluate for point sets A and B their smallest distance $\min_{f \in \mathcal{F}} \delta(f(A), B)$ under transformations $\mathcal{F}$ or to test if this distance is $\leq \varepsilon$. This problem is called the *distance evaluation problem*.

Key Results

A folk theorem is a voting algorithm to solve EPPM under translations in $O(|P||T| \log(|T||P|))$ time: Collect all translations mapping each point of P to each point of T, sort the set, and report the translation getting most votes. If some translation gets $|P|$ votes, then a subset I such $f(P) = I$ is found. With some care in organizing the sorting, one can achieve $O(|P||T| \log |P|)$ time [13].

The voting algorithm also solves the LCP problem under translations. A faster algorithm specific to EPPM is as follows: Let $p_1, p_2, \cdots p_m$ and $t_1, t_2, \cdots t_n$ be the lists of pattern and target points, respectively, *lexicographicly ordered* according to their d-dimensional coordinate values. Consider the translation $f_{i_1} = t_{i_1} - p_1$, for any $1 \leq i_1 \leq n$. One can scan the target points in the lexicographic order to find a point t_{i_2} such that $p_2 + f_{i_1} = t_{i_2}$. If such is found, one can continue scanning from t_{i_2+1} on to find t_{i_3} such that $p_3 + f_{i_1} = t_{i_3}$. This process is continued until a translated point of P does not occur in T or until a translated occurrence of the entire P is found. Careful implementation of this idea leads to the following result showing that the time bound of the naive string matching algorithm is possible also for the exact point pattern matching under translations.

Theorem 1 (Ukkonen et al. 2003 [13]) *The EPPM problem under translations for point pattern P and target T can be solved in $O(mn)$ time and $O(n)$ space where $m = |P| \leq |T| = n$.*

Quadratic running times are probably the best one can achieve for PPM algorithms:

Theorem 2 (Clifford et al. 2006 [10]) *The LCP problem under translations is* 3SUM*-hard.*

This means that an $o(|P||T|)$ time algorithm for LCP would yield an $o(n^2)$ algorithm for the 3SUM problem, where $|T| = n$ and $|P| = \Theta(n)$. The 3SUM problem asks, given n numbers, whether there are three numbers a, b, and c among them such that $a + b + c = 0$; finding a subquadratic algorithm for 3SUM would be a surprise [5]. For a more in-depth combinatorial characterization of the geometric properties of the EPPM problem, see [7].

For the distance evaluation problems there are plethora of results. An excellent survey of the key results until 1999 is by Alt and Guibas [2]. As an example, consider in the 2-dimensional case how one can decide in $O(n \log n)$ time whether there is a transformation f composed of translation, rotation and scale, such that $f(A) = B$, where $A, B \subset \mathbb{R}^2$ and $n = |A| = |B|$: The idea is to convert A and B into an invariant form such that one can easily check their congruence under the transformations. First, scale is taken into account by scaling A to have the same *diameter* as B (in $O(n \log n)$ time). If A and B are congruent, then they must have the same *centroids* (which can be computed $O(n)$ time). Consider rotating a line from the centroid and listing the angles and distances to other points in the order they are met during the rotation. Having done this (in $O(n \log n)$ time) on both A and B, the lists of angles and distances should be *cyclic shifts* of each other; the list L_A of A occurs as a substring in $L_B L_B$, where L_B is the list of B. This latter step can be done in $O(n)$ time using any linear time exact string matching algorithm. One obtains the following result.

Theorem 3 (Atkinson 1987 [4]) *It is possible to decide in $O(n \log n)$ time whether there is a transformation f com-*

posed of translation, rotation and scale, such that $f(A) = B$, where $A, B \subset \mathbb{R}^2$ and $|A| = |B| = n$.

Approximate variant of the above problem is much harder. Denote by $f(A) =^{\varepsilon} B$ the *directed approximate congruence* of point sets A and B, meaning that there is a one-to-one mapping from $f(A)$ to B such that for each point in $f(A)$ its image in B is ε-close. The following result demonstrates the added difficulty.

Theorem 4 (Alt et al. 1988 [3]) *It is possible to decide in $O(n^6)$ time whether there is a translation f such that $f(A) =^{\varepsilon} B$, where $A, B \subset \mathbb{R}^2$ and $|A| = |B| = n$. The same algorithm solves the corresponding LACP problem for point pattern P and target T under the one-to-one matching condition in $O((mn)^3)$ time, where $m = |P| \le |T| = n$.*

To get an idea of the techniques to achieve the $O((mn)^3)$ time algorithm for LACP, consider first the one-dimensional version, i. e. let $P, T \subset \mathbb{R}$. Observe, that if there is a translation f' such that $f'(P) =^{\varepsilon} T$, then there is a translation f such that $f(P) =^{\varepsilon} T$ *and* a point $p \in P$ that is mapped *exactly* at ε-distance of a point $t \in T$. This lets one concentrate on these $2mn$ *representative translations*. Consider these translations sorted from left to right. Denote the left-most translation by f. Create a bipartite graph, whose nodes are the points in P and in T on the different parties. There is an edge between $p \in P$ and $t \in T$ if and only if $f(p)$ is ε-close to t. Finding a maximum matching in this graph tells the size of the largest approximately common point set after applying the translation f. One can repeat this on each representative translation to find the overall largest common point set. When the representative translations are considered from left to right, the bipartite graph instances are such that one can compute the maximum matchings greedily at each translation in time $O(|P|)$ [6]. Hence, the algorithm solves the one-dimensional LACP problem under translations and one-to-one matching condition in time $O(m^2 n)$, where $m = |P| \le |T| = n$.

In the two-dimensional case, the set of representative translations is more implicitly defined: In short, the mapping of each point $p \in P$ ε-close to each point $t \in T$, gives mn circles. The boundary of each such circle is partitioned into intervals such that the end points of these intervals can be chosen as representative translations. There are $O((mn)^2)$ such representative translations. As in the one-dimensional case, each representative translation defines a bipartite graph. Once the representative translations along a circle are processed e. g. counterclockwise, the bipartite graph changes only by one edge at a time. This allows an $O(mn)$ time update for the maximum matching at each representative translation yielding an overall $O((mn)^3)$ time algorithm [3].

More efficient algorithms for variants of this problem have been developed by Efrat, Itai, and Katz [11], as by-products of more efficient bipartite matching algorithms for points on a plane. Their main result is the following:

Theorem 5 (Efrat et al. 2001 [11]) *It is possible to decide in $O(n^5 \log n)$ time whether there is a translation f such that $f(A) =^{\varepsilon} B$, where $A, B \subset \mathbb{R}^2$ and $|A| = |B| = n$.*

The problem becomes somewhat easier when the one-to-one matching condition is relaxed; one-to-one condition seems to necessitate the use of bipartite matching in one form or another. Without the condition, one can match the points independently of each other. This gives many tools to preprocess and manipulate the point sets during the algorithm using dynamic geometric data structures. Such techniques are exploited e. g. in the following result.

Theorem 6 (Chew and Kedem 1992 [8]) *The LACP problem under translations and using directed Hausdorff distance and the L_1 norm, can be solved in $O(mn \log n)$ time, where $P, T \subset \mathbb{R}^2$ and $m = |P| \le |T| = n$. The distance evaluation problem for directed Hausdorff distance can be solved in $O(n^2 \log^2 n)$ time.*

Most algorithms revisited here have relatively high running times. To obtain faster algorithms, it seems that randomization and approximation techniques are necessary. See [9] for a comprehensive summary of the main achievements in that line of development.

Finally, note that the linear transformations considered here are not always enough to model a real-world problem – even when approximate congruence is allowed. Sometimes the proper transformation between two point sets (or between their subsets) is non-linear, without an easily parametrizable representation. Unfortunately, the formulations trying to capture such non-uniformness have been proven NP-hard [1] or even NP-hard to approximate within any constant factor [12].

Applications

Point pattern matching is a fundamental problem that naturally arises in many application domains such as computer vision, pattern recognition, image retrieval, music information retrieval, bioinformatics, dendrochronology, and many others.

Cross References

- ► Assignment Problem
- ► Multidimensional String Matching

► Sequential Exact String Matching
► Stable Matching

Recommended Reading

1. Akutsu, T., Kanaya, K., Ohyama, A., Fujiyama, A.: Point matching under non-uniform distortions. Discret. Appl. Math. **127**, 5–21 (2003)
2. Alt, H., Guibas, L.: Discrete geometric shapes: Matching, interpolation, and approximation. In: Sack, J.R., Urrutia, J. (eds.) Handbook of Computational Geometry, pp. 121–153. Elsevier Science Publishers B.V. North-Holland, Amsterdam (1999)
3. Alt, H., Mehlhorn, K., Wagener, H., Welzl, E.: Congruence, similarity and symmetries of geometric objects. Discret. Comput. Geom. **3**, 237–256 (1988)
4. Atkinson, M.D.: An optimal algorithm for geometric congruence. J. Algorithms **8**, 159–172 (1997)
5. Barequet, G., Har-Peled, S.: Polygon containment and translational min-hausdorff-distance between segment sets are 3SUM-hard. Int. J. Comput. Geom. Appl. **11**(4), 465–474 (2001)
6. Böcker, S., Mäkinen, V.: Maximum line-pair stabbing problem and its variations. In: Proc. 21st European Workshop on Computational Geometry (EWCG'05), pp. 183–186. Technische Universität Eindhoven, The Netherlands (2005)
7. Brass, P., Pach, J.: Problems and results on geometric patterns. In: Avis, D. et al. (eds.) Graph Theory and Combinatorial Optimization, pp. 17–36. Springer Science + Business Media Inc., NY, USA (2005)
8. Chew, L.P., Kedem, K.: Improvements on geometric pattern matching problems. In: Proc. Scandinavian Workshop Algorithm Theory (SWAT). LNCS, vol. 621, pp. 318–325. Springer, Berlin (1992)
9. Choi, V., Goyal, N.: An efficient approximation algorithm for point pattern matching under noise. In: Proc. 7th Latin American Symposium on Theoretical Informatics (LATIN 2006). LNCS, vol. 3882, pp. 298–310. Springer, Berlin (2006)
10. Clifford, R., Christodoukalis, M., Crawford, T., Meredith, D., Wiggins, G.: A Fast, Randomised, Maximum Subset Matching Algorithm for Document-Level Music Retrieval. In: Proc. International Conference on Music Information Retrieval (ISMIR 2006), University of Victoria, Canada (2006)
11. Efrat, A., Itai, A., Katz, M.: Geometry Helps in Bottleneck Matching and Related Problems. Algorithmica **31**(1), 1–28 (2001)
12. Mäkinen, V., Ukkonen, E.: Local Similarity Based Point-Pattern Matching. In: Proc. 13th Annual Symposium on Combinatorial Pattern Matching (CPM 2002). LNCS, vol. 2373, pp. 115–132. Springer, Berlin (2002)
13. Ukkonen, E., Lemström, K., Mäkinen, V.: Sweepline the music! In: Klein, R. Six, H.W., Wegner, L. (eds.) Computer Science in Perspective, Essays Dedicated to Thomas Ottmann. LNCS, vol. 2598, pp. 330–342. Springer (2003)

Position Auction

2005; Varian

Aries Wei Sun
Department of Computer Science, City University of Hong Kong, Hong Kong, China

Keywords and Synonyms

Adword auction

Problem Definition

This problem is concerned with the Nash equilibria of a game based on the ad auction used by Google and Yahoo. This research work [5] is motivated by the huge revenue that the adword auction derives every year. It defines two types of Nash equilibrium in the position auction game, applies economic analysis to the equilibria, and provides some empirical evidence that the Nash equilibria of the position auction describes the basic properties of the prices observed in Google's adword auction reasonably accurately. The problem being studied is closely related to the assignment game studied by [4,1,3]. And [2] has independently examined the problem and developed related results.

The Model and its Notations

Consider the problem of assigning agents $a = 1, 2, \dots, A$ to slots $s = 1, 2, \dots, S$ where agent a's valuation for slot s is given by $u_{as} = v_a x_s$. The slots are numbered such that $x_1 > x_2 > \dots > x_S$. It is assumed that $x_S = 0$ for all $s > S$ and the number of agents is greater than the number of slots. A higher position receives more clicks, so x_s can be interpreted as the click-through rate for slot s. The value $v_a > 0$ can be interpreted as the expected profit per click so $u_{as} = v_a x_s$ indicates the expected profit to advertiser a from appearing in slot s.

The slots are sold via an auction. Each agent bids an amount b_a, with the slot with the best click through rate being assigned to the agent with the highest bid, the second-best slot to the agent with the second highest bid, and so on. Renumbering the agents if necessary, let v_s be the value per click of the agent assigned to slot s. The price agent s faces is the bid of the agent immediately below him, so $p_t = b_{t+1}$. Hence the net profit that agent a can expect to make if he acquires slot s is $(v_a - p_s)\, x_s = (v_a - b_{s+1})\, x_s$.

Definitions

Definition 1 A Nash equilibrium set of prices *(NE)* satisfies

$$\begin{aligned}(v_s - p_s)\, x_s &\geq (v_s - p_t)\, x_t, \quad \text{for } t > s \\ (v_s - p_s)\, x_s &\geq (v_s - p_{t-1})\, x_t, \quad \text{for } t < s\end{aligned}$$

where $p_t = b_{t+1}$.

Definition 2 A symmetric Nash equilibrium set of prices *(SNE)* satisfies

$$\left(v_s - p_s\right) x_s \geq \left(v_s - p_t\right) x_t \text{ for all } t \text{ and } s.$$

Equivalently,

$$v_s\left(x_s - x_t\right) \geq p_s x_s - p_t x_t \text{ for all } t \text{ and } s.$$

Key Results

Facts of NE and SNE

Fact 1 (Non-negative surplus) In an SNE, $v_s \geq p_s$.
Fact 2 (Monotone values) In an SNE, $v_{s-1} \geq v_s$, for all s.
Fact 3 (Monotone prices) In an SNE, $p_{s-1}x_{s-1} > p_s x_s$ and $p_{s-1} \geq p_s$ for all s. If $v_s > p_s$ then $p_{s-1} > p_s$.
Fact 4 ($NE \supset SNE$) If a set of prices is an SNE then it is an NE.
Fact 5 (One-step solution) If a set of bids satisfies the symmetric Nash equilibria inequalities for $s + 1$ and $s - 1$, then it satisfies these inequalities for all s.
Fact 6 The maximum revenue NE yields the same revenue as the upper recursive solution to the SNE.

A Sufficient and Necessary Condition of the Existence of a Pure Strategy Nash Equilibrium in the Position Auction Game

Theorem 1 *In the position auction described before, a pure strategy Nash equilibrium exists if and only if all the intervals*

$$\left[\frac{p_s x_s - p_{s+1}x_{s+1}}{x_s - x_{s+1}}, \frac{p_{s-1}x_{s-1} - p_s x_s}{x_{s-1} - x_s}\right], \text{ for } s = 2, 3, \ldots, S$$

are non-empty.

Applications

The model studied in this paper is a simple and elegant abstraction of the real adword auctions used by search engines such as Google and Yahoo. Different search engines have slightly different rules. For example, Yahoo ranks the advertisers according to their bids, while Google ranks the advertisers not only according to their bids but also according to the likelihood of their links being clicked.

However, similar analysis can be applied to real world situations, as the author has demonstrated above for the Google adword auction case.

Cross References

► Adwords Pricing

Recommended Reading

1. Demange, G., Gale, D., Sotomayor, M.: Multi-item auctions. J. Polit. Econ. **94**(4), 863–72 (1986)
2. Edelman, B., Ostrovsky, M., Schwartz, M.: Internet advertising and the generalized second price auction. NBER Working Paper, 11765, November 2005
3. Roth, A., Sotomayor, M.: Two-Sided Matching. Cambridge University Press, Cambridge (1990)
4. Shapely, L., Shubik, M.: The Assignment Game I: the core. Int. J. Game Theor. **1**, 111–130 (1972)
5. Varian, H.R.: Position auctions. Int. J. Ind. Organ. **25**(6), 1163–1178 (2007)

Predecessor Search

2006; Pătraşcu, Thorup

Mihai Pătraşcu
CSAIL, MIT, Cambridge, MA, USA

Keywords and Synonyms

Predecessor problem Successor problem IP lookup

Problem Definition

Consider an ordered universe U, and a set $T \subset U$ with $|T| = n$. The goal is to preprocess T, such that the following query can be answered efficiently: given $x \in U$, report the predecessor of x, i. e. $\max\{y \in T \mid y < x\}$. One can also consider the dynamic problem, where elements are inserted and deleted into T. Let t_{q} be the query time, and t_{u} the update time.

This is a fundamental search problem, with an impressive number of applications. Later, this entry discusses IP lookup (forwarding packets on the Internet), orthogonal range queries and persistent data structures as examples.

The problem was considered in many computational models. In fact, most models below were initially defined to study the predecessor problem.

Comparison model: The problem can be solved through binary search in $\Theta(\lg n)$ comparisons. There is a lot of work on adaptive bounds, which may be sublogarithmic. Such bounds may depend on the finger distance, the working set, entropy etc.
Binary search trees: Predecessor search is one of the fundamental motivations for binary search trees. In this restrictive model, one can hope for an instance optimal (competitive) algorithm. Attempts to achieve this are described in a separate entry.[1]

[1] $O(\log\log n)$*-competitive Binary Search Trees* (2004; Demaine, Harmon, Iacono, Pătraşcu)

Word RAM: Memory is organized as words of b bits, and can be accessed through indirection. Constant-time operations include the standard operations in a language such as C (addition, multiplication, shifts and bitwise operations).

It is standard to assume the universe is $U = \{1, \dots, 2^{\ell}\}$, i. e. one deals with ℓ-bit integers. The floating point representation was designed so that order is preserved when values are interpreted as integers, so any algorithm will also work for ℓ-bit floating point numbers.

The standard *transdichotomous* assumption is that $b = \ell$, so that an input integer is represented in a word. This implies $b \geq \lg n$.

Cell-probe model: This is a nonuniform model stronger than the word RAM, in which the operations are arbitrary functions on the memory words (cells) which have already been probed. Thus, t_{q} only counts the number of cell probes. This is an ideal model for lower bounds, since it does not depend on the operations implemented by a particular computer.

Communication games: Let Alice have the query x, and Bob have the set T. They are trying to find the predecessor of x through τ rounds of communication, where in each round Alice sends m_{A} bits, and Bob replies with m_{B} bits.

This can simulate the cell-probe model when $m_{\mathrm{B}} = b$ and m_{A} is the logarithm of the memory size. Then $\tau \leq t_{\mathrm{q}}$ and one can use communication complexity to obtain cell-probe lower bounds.

External memory: The unit of access is a page, containing B words of ℓ bits each. B-trees solve the problem with query and update time $O(\log_B n)$, and one can also achieve this oblivious to the value of B.[2] The cell-probe model with $b = B \cdot \ell$ is stronger than this model.

***AC*^{0} RAM:** This is a variant of the word RAM in which allowable operations are functions that have constant depth, unbounded fan-in circuits. This excludes multiplication from the standard set of operations.

RAMBO: this is a variant of the RAM with a nonstandard memory, where words of memory can overlap in their bits. In the static case this is essentially equivalent to a normal RAM. However, in the dynamic case updates can be faster due to the word overlap [5].

The worst-case logarithmic bound for comparison search is not particularly informative when efficiency really matters. In practice, B-trees and variants are standard when dealing with huge data sets. Solutions based on RAM tricks are essential when the data set is not too large, but a fast query time is crucial, such as in software solutions to IP lookup [7].

[2]See *Cache-oblivious B-tree* (2005; Bender, Demaine, Farach-Colton).

Key Results

Building on a long line of research, Pătraşcu and Thorup [15,16] finally obtained matching upper and lower bounds for the static problem in the word RAM, cell-probe, external memory and communication game models.

Let S be the number of words of space available. (In external memory, this is equivalent to S/B pages.) Define $a = \lg S \cdot \ell / n$. Also define $\lg x = \lceil \log_2(x+2) \rceil$, so that $\lg x \geq 1$ even if $x \in [0, 1]$. Then the optimal search time is, up to constant factors:

$$\min \begin{cases} \log_b n = \Theta(\min\{\log_B n, \log_{\ell} n\}) \\ \lg \frac{\ell - \lg n}{a} \\ \dfrac{\lg \frac{\ell}{a}}{\lg\left(\frac{a}{\lg n} \cdot \lg \frac{\ell}{a}\right)} \\ \dfrac{\lg \frac{\ell}{a}}{\lg\left(\dfrac{\lg \frac{\ell}{a}}{\lg \frac{\lg n}{a}}\right)} \end{cases} \tag{1}$$

The bound is achieved by a deterministic query algorithm. For any space S, the data structure can be constructed in time $O(S)$ by a randomized algorithm, starting with the set T given in sorted order. Updates are supported in expected time $t_{\mathrm{q}} + O(S/n)$. Thus, besides locating the element through one predecessor query, updates change a minimal fraction of the data structure.

Lower bounds hold in the powerful cell-probe model, and hold even for randomized algorithms. When $S \geq n^{1+\varepsilon}$, the optimal trade-off for communication games coincides to (1). Note that the case $S = n^{1+o(1)}$ essentially disappears in the reduction to communication complexity, because Alice's messages only depends on $\lg S$. Thus, there is no asymptotic difference between $S = O(n)$ and, say, $S = O(n^2)$.

Upper Bounds

The following algorithmic techniques give the optimal result:

- *B-trees* give $O(\log_B n)$ query time with linear space.
- *Fusion trees*, by Fredman and Willard [10], achieve a query time of $O(\log_b n)$. The basis of this is a *fusion node*, a structure which can search among b^{ε} values in constant time. This is done by recognizing that

only a few bits of each value are essential, and packing the relevant information about all values in a single word.

- *Van Emde Boas search* [18] can solve the problem in $O(\lg \ell)$ time by binary searching for the length of the longest common prefix between the query and a value in T. Beginning the search with a table lookup based on the first $\lg n$ bits, and ending when there is enough space to store all answers, the query time is reduced to $O(\lg((\ell - \lg n)/a))$.
- A technique by *Beame and Fich* [4] can perform a multiway search for the longest common prefix, by maintaining a careful balance between ℓ and n. This is relevant when the space is at least $n^{1+\varepsilon}$, and gives the third branch of (1). Pătraşcu and Thorup [15] show how related ideas can be implemented with smaller space, yielding the last branch of (1).

Observe that external memory only features in the optimal trade-off through the $O(\log_B n)$ term coming from B-trees. Thus, it is optimal to either use the standard, comparison-based B-trees, or use the best word RAM strategy which completely ignores external memory.

Lower Bounds

All lower bounds before [15] where shown in the communication game model. Ajtai [1] was the first to prove a superconstant lower bound. His results, with a correction by Miltersen [12], show that for polynomial space, there exists n as a function of ℓ making the query time $\Omega(\sqrt{\lg \ell})$, and likewise there exists ℓ a function of n making the query complexity $\Omega(\sqrt[3]{\lg n})$.

Miltersen et al [13] revisited Ajtai's proof, extending it to randomized algorithms. More importantly, they captured the essence of the proof in an independent *round elimination lemma*, which is an important tool for proving lower bounds in asymmetric communication.

Beame and Fich [4] improved Ajtai's lower bounds to $\Omega(\lg \ell / \lg \lg \ell)$ and $\Omega(\sqrt{\lg n / \lg \lg n})$ respectively. Sen and Venkatesh [17] later gave an improved round elimination lemma, which can reprove these lower bounds, but also for randomized algorithms.

Finally, using the message compression lemma of [6] (an alternative to round elimination), Pătraşcu and Thorup [15] showed an optimal trade-off for communication games. This is also an optimal lower bound in the other models when $S \geq n^{1+\varepsilon}$, but not for smaller space.

More importantly, [15] developed the first tools for proving lower bounds exceeding communication complexity, when $S = n^{1+o(1)}$. This showed the first separation ever between a data structure or polynomial size, and one of near linear size. This is fundamentally impossible through a direct communication lower bound, since the reduction to communication games only depends on $\lg S$.

The full result of Pătraşcu and Thorup [15] it the trade-off (1). Initially, this was shown only for deterministic query algorithms, but eventually it was extended to a randomized lower bound as well [16]. Among the surprising consequences of this result was that the classic van Emde Boas search is optimal for near-linear space (and thus for dynamic data structures), whereas with quadratic space it can be beaten by the technique of Beame and Fich.

A key technical idea of [15] is to analyze many queries simultaneously. Then, one considers a communication game involving all queries, and proves a direct-sum version of the round elimination lemma. Arguably, the proof is even simpler than for the regular round elimination lemma. This is achieved by considering a stronger model for the inductive analysis, in which the algorithm is allowed to *reject* a large fraction of the queries before starting to make probes.

Bucketing

The rich recursive structure of the problem can not only be used for fast queries, but also to optimize the space and update time – of course, within the limits of (1). The idea is to place ranges of consecutive values in buckets, and include a single representative of each bucket in the predecessor structure. After performing a query on the predecessor structure (now with fewer elements), one need only search within the relevant bucket.

Because buckets of size $w^{O(1)}$ can be handled in constant time by fusion trees, it follows that factors of w in space are irrelevant. A more extreme application of the idea is given by *exponential trees* [3]. Here buckets have size $\Theta(n^{1-\gamma})$, where γ is a sufficiently small constant. Buckets are handled recursively in the same way, leading to $O(\lg \lg n)$ levels. If the initial query time is at least $t_q \geq \lg^{\varepsilon} n$, the query times at each level decrease geometrically, so overall time only grows by a constant factor. However, any polynomial space is reduced to linear, for an appropriate choice of γ. Also, the exponential tree can be updated in $O(t_q)$ time, even if the original data structure was static.

Applications

Perhaps the most important application of predecessor search is IP lookup. This is the problem solved by routers for each packet on the Internet, when deciding which subnetwork to forward the packet to. Thus, it is probably the most run algorithmic problem in the world. Formally, this is an *interval stabbing* query, which is equivalent to predecessor search in the static case [9]. As this is a problem where efficiency really matters, it is important to note that the fastest deployed software solutions [7] use integer search strategies (not comparison-based), as theoretical results would predict.

In addition, predecessor search is used pervasively in data structures, when reducing problems to *rank space*. Given a set T, one often wants to relabel it to the simpler $\{1, \dots, n\}$ ("rank space"), while maintaining order relations. If one is presented with new values dynamically, the need for a predecessor query arises. Here are a couple of illustrative examples:

- In orthogonal range queries, one maintains a set of points in U^d, and queries for points in some rectangle $[a_1, b_1] \times \cdots \times [a_d, b_d]$. Though bounds typically grow exponentially with the dimension, the dependence on the universe can be factored out. At query time, one first runs $2d$ predecessor queries transforming the universe to $\{1, \dots, n\}^d$.
- To make pointer data structures persistent [8], an outgoing link is replaced by a vector of pointers, each valid for some period of time. Deciding which link to follow (given the time being queried) is a predecessor problem.

Finally, it is interesting to note that the lower bounds for predecessor hold, by reductions, for all applications described above. To make these reductions possible, the lower bounds are in fact shown for the weaker *colored predecessor* problem. In this problem, the values in T are colored red or blue, and the query only needs to find the color of the predecessor.

Open Problems

It is known [2] how to implement fusion trees with AC^0 instructions, but not the other query strategies. What is the best query trade-off achievable on the AC^0 RAM? In particular, can van Emde Boas search be implemented with AC^0 instructions?

For the dynamic problem, can the update times be made deterministic? In particular, can van Emde Boas search be implemented with fast deterministic updates? This is a very appealing problem, with applications to deterministic dictionaries [14]. Also, can fusion nodes be updated deterministically in constant time? Atomic heaps [11] achieve this when searching only among $(\lg n)^\varepsilon$ elements, not b^ε.

Finally, does an update to the predecessor structure require a query? In other words, can $t_u = o(t_q)$ be obtained, while still maintaining efficient query times?

Cross References

► Cache-Oblivious B-Tree
► $O(\log\log n)$-competitive Binary Search Tree

Recommended Reading

1. Ajtai, M.: A lower bound for finding predecessors in Yao's cell probe model. Combinatorica **8**(3), 235–247 (1988)
2. Andersson, A., Miltersen, P.B., Thorup, M.: Fusion trees can be implemented with AC^0 instructions only. Theor. Comput. Sci. **215**(1–2), 337–344 (1999)
3. Andersson, A., Thorup, M.: Dynamic ordered sets with exponential search trees. CoRR cs.DS/0210006. See also FOCS'96, STOC'00, 2002
4. Beame, P., Fich, F.E.: Optimal bounds for the predecessor problem and related problems. J. Comput. Syst. Sci. **65**(1), 38–72 (2002). See also STOC'99
5. Brodnik, A., Carlsson, S., Fredman, M.L., Karlsson, J., Munro, J.I.: Worst case constant time priority queue. J. Syst. Softw. **78**(3), 249–256 (2005). See also SODA'01
6. Chakrabarti, A., Regev, O.: An optimal randomised cell probe lower bound for approximate nearest neighbour searching. In: Proc. 45th IEEE Symposium on Foundations of Computer Science (FOCS), 2004, pp. 473–482
7. Degermark, M., Brodnik, A., Carlsson, S., Pink, S.: Small forwarding tables for fast routing lookups. In: Proc. ACM SIGCOMM, 1997, pp. 3–14
8. Driscoll, J.R., Sarnak, N., Sleator, D.D., Tarjan, R.E.: Making data structures persistent. J. Comput. Syst. Sci. **38**(1), 86–124 (1989). See also STOC'86
9. Feldmann, A., Muthukrishnan, S.: Tradeoffs for packet classification. In: Proc. IEEE INFOCOM, 2000, pp. 1193–1202
10. Fredman, M.L., Willard, D.E.: Surpassing the information theoretic bound with fusion trees. J. Comput. Syst. Sci. **47**(3), 424–436 (1993). See also STOC'90
11. Fredman, M.L., Willard, D.E.: Trans-dichotomous algorithms for minimum spanning trees and shortest paths. J. Comput. Syst. Sci. **48**(3), 533–551 (1994). See also FOCS'90
12. Miltersen, P.B.: Lower bounds for Union-Split-Find related problems on random access machines. In: 26th ACM Symposium on Theory of Computing (STOC), 1994, pp. 625–634
13. Miltersen, P.B., Nisan, N., Safra, S., Wigderson, A.: On data structures and asymmetric communication complexity. J. Comput. Syst. Sci. **57**(1), 37–49 (1998). See also STOC'95
14. Pagh, R.: A trade-off for worst-case efficient dictionaries. Nord. J. Comput. **7**, 151–163 (2000). See also SWAT'00
15. Pătraşcu, M., Thorup, M.: Time-space trade-offs for predecessor search. In: Proc. 38th ACM Symposium on Theory of Computing (STOC), 2006, pp. 232–240
16. Pătraşcu, M., Thorup, M.: Randomization does not help searching predecessors. In: Proc. 18th ACM/SIAM Symposium on Discrete Algorithms (SODA), 2007
17. Sen, P., Venkatesh, S.: Lower bounds for predecessor searching in the cell probe model. arXiv:cs.CC/0309033. See also ICALP'01, CCC'03, 2003
18. van Emde Boas, P., Kaas, R., Zijlstra, E.: Design and implementation of an efficient priority queue. Math. Syst. Theor. **10**, 99–127 (1977). Announced by van Emde Boas alone at FOCS'75

Price of Anarchy

2005; Koutsoupias

George Christodoulou
Max-Planck-Institute for Computer Science,
Saarbruecken, Germany

Keywords and Synonyms

Coordination ratio

Problem Definition

The *Price of Anarchy*, captures the lack of coordination in systems where users are selfish and may have conflicted interests. It was first proposed by Koutsoupias and Papadimitriou in [9], where the term *coordination ratio* was used instead, but later Papadimitriou in [12] coined the term Price of Anarchy, that finally prevailed in the literature.

Roughly, the Price of Anarchy is the system cost (e. g. makespan, average latency) of the worst-case Nash Equilibrium over the optimal system cost, that would be achieved if the players were forced to coordinate. Although it was originally defined in order to analyze a simple load-balancing game, it was soon applied to numerous variants and to more general games. The family of *(weighted) congestion games* [11,13] is a nice abstract form to describe most of the alternative settings.

The Price of Anarchy may vary, depending on the

- equilibrium solution concept *(e. g. pure, mixed, correlated equilibria)*
- characteristics of the congestion game
  - Players Set *(e. g. atomic – non atomic)*
  - Strategy Set *(e. g. symmetric-asymmetric, parallel machines-network-general)*
  - Utility *(e. g. linear, polynomial)*
- social cost *(e. g. maximum, sum, total latency)*.

Notations

Let G be a (finite) game, that is determined by the triple $(N, (S_i)_{i\in N}, (c_i)_{i\in N})$. $N = \{1, \dots, n\}$ is the set of the players, that participate in the game. S_i is a *pure strategy set* for player i. An element $A_i \in S_i$ is a *pure strategy* for player $i \in N$. A *pure strategy profile* $A = (A_1, \dots, A_n)$ is a vector of pure strategies, one for each player. The set of all possible pure strategy profiles is denoted by $S = S_1 \times \dots \times S_n$. The *cost* of a player $i \in N$, for a pure strategy, is determined by a cost function $c_i : S \mapsto \mathbb{R}$.

A pure strategy profile A is a *pure Nash equilibrium*, if none of the players $i \in N$ can benefit, by unilaterally deviating to another pure strategy $s_i \in S_i$:

$$c_i(A) \le c_i(A_{-i}, s_i) \qquad \forall i \in N, \quad \forall s_i \in S_i \ ,$$

where (A_{-i},s_i) is the simple strategy profile that results when just the player i deviates from strategy $A_i \in S_i$ to strategy $s_i \in S_i$.

A *mixed strategy* p_i for a player $i \in N$, is a probability distribution over her pure strategy set S_i. A mixed strategy profile p is the tuple $p = (p_1, \dots p_n)$, where player i chooses mixed strategy p_i. The expected cost of a player $i \in N$ with respect to the p, is

$$c_i(p) = \sum_{A\in S} p(A) c_i(A) \ ,$$

where $p(A) = \prod_{i\in N} p_i(A_i)$ is the probability that pure strategy A occurs, with respect to $(p_i)_{i\in N}$. A mixed strategy profile p is a *Nash Equilibrium*, if and only if

$$c_i(p) \le c_i(p_{-i}, s_i) \quad \forall i \in N, \ \forall s_i \in S_i \ .$$

The *social cost* of a pure strategy profile A, denoted by $\mathrm{SC}(A)$, is the maximum cost of a player $\mathrm{MAX}(A) = \max_{i\in N} c_i(A)$ or the average cost of a player. For simplicity, the sum of the players cost is considered (that is n times the average cost) $\mathrm{SUM}(A) = \sum_{i\in N} c_i(A)$. The same definitions extend naturally for the case of mixed strategies, but with expected costs in this case.

The (mixed) *Price of Anarchy* [9] for a game, is the worst-case ratio, among all the (mixed) Nash Equilibria, of the social cost over the optimal cost, $\mathrm{OPT} = \min_{P\in S} \mathrm{SC}(P)$.

$$\mathrm{PA} = \max_{p \text{ is N.E.}} \frac{\mathrm{SC}(p)}{\mathrm{OPT}} \ .$$

The Price of Anarchy for a class of games, is the maximum (supremum) price of anarchy among all the games of this class.

Congestion Games Here, a general class of games is described, that contains most of the games for which Price of Anarchy is studied in the literature. A *congestion game* [11,13], is defined by the tuple $(N, E, (S_i)_{i\in N}, (f_e)_{e\in E})$, where $N = \{1, \dots, n\}$ is a set of players, E is a set of *facilities*, $S_i \subseteq 2^E$ is the pure strategy set for player i; a pure strategy $A_i \in S_i$ is a subset of the facility set, and f_e is a *cost (or latency) function*[1] with respect to the facility $e \in E$.

[1]Unless otherwise stated, linear cost functions are considered throughout this article. For additional results on more general cost functions see entries ► Best Response Algorithms for Selfish Routing, ► Computing Pure Equilibria in the Game of Parallel Links, ► Price of Anarchy for Routing on Parallel Links

A pure strategy profile $A = (A_1, \dots, A_n)$ is a vector of pure strategies, one for each player. The cost $c_i(A)$ of player i for the pure strategy profile A is given by

$$c_i(A) = \sum_{e \in A_i} f_e(n_e(A)) ,$$

where $n_e(A)$ is the number of the players that use facility e in A.

In general games, a pure Nash equilibrium may not exist. Rosenthal [13] showed that every congestion game possess at least a pure Nash equilibrium. In particular he defined a potential function over the strategy space

$$\Phi(A) = \sum_{e \in E} \sum_{i=1}^{n_e(A)} f_e(i) .$$

He proved that every local optimum of this potential function is a pure Nash Equilibrium.

$$\Phi(A) \leq \Phi(A_{-i}, s_i), \quad \forall i \in N,\ s_i \in S_i .$$

A congestion game is called *symmetric or single-commodity*, if all the players have the same strategy set: $S_i = C$. The term *asymmetric or multi-commodity* is used, to refer to all the games including the symmetric ones.

A special class of congestion games is the class of *network congestion games*. In this games, the facilities are edges of a (multi)graph $G(V,E)$. The pure strategy set for a player $i \in N$ is the simple paths set from a source $s_i \in V$ to a destination $t_i \in V$. In network symmetric congestion games, all the players have the same source and destination.

A natural generalization of congestion games are the *weighted congestion games*, where every player controls an amount of traffic w_i. The cost of each facility $e \in E$ depends on the total load $\theta_e(A)$ of the facility. In this case, an additional social cost function makes sense, i. e. *total latency*. For a pure strategy profile $A \in S$, the total latency is defined as a weighted sum

$$C(A) = \sum_{e \in E} \theta_e(A) \cdot f_e(\theta_e(A)) .$$

Notice that the sum and the total latency coincide for the case of unweighted congestion games.

In a congestion game with *splittable weights (divisible demands)*, every player $i \in N$, instead of fixing a single pure strategy, she is allowed to distribute her demand among her pure strategy set.

In a *non-atomic congestion game*, there are k different player types $1 \dots k$. Players are infinitesimal and for each player type i the continuum of the players is denoted by the interval $[0, n_i]$. In general, each player type contributes in a different way to the congestion on the facility $e \in E$, and this contribution is determined by a positive *rate of consumption* $r_{s,e}$ for a strategy $s \in S_i$ and a facility $e \in s$. Each player chooses a strategy that results in a *strategy distribution* $x = (x_s)_{s \in S}$, with $\sum_{s \in S_i} x_s = n_i$.

Key Results

Maximum Social Cost

Here, it is considered the price of anarchy in the case where the social cost is the maximum cost among the players. Formally, for a pure strategy profile A, the social cost is

$$\mathrm{SC}(A) = \textsc{Max}(A) = \max_{i \in N} c_i(A) .$$

The definition naturally extends to mixed strategies.

Theorem 1 ([7,8,9,10]) *The price of anarchy for m identical machines is* $\Theta(\log m / \log\log m)$.

Theorem 2 ([7]) *The price of anarchy for m machines with speeds* $s_1 \geq s_2 \geq \dots \geq s_m$ *is*

$$\Theta\left(\min\left\{\frac{\log m}{\log\log\log m}, \frac{\log m}{\log\left(\frac{\log m}{\log(s_1/s_m)}\right)}\right\}\right) .$$

Theorem 3 ([3]) *The price of anarchy for m identical machines, in the asymmetric case is* $\Theta(\log m / \log\log m)$ *for pure equilibria and* $\Theta(\log m / \log\log\log m)$ *for mixed equilibria.*

Theorem 4 ([5]) *The price of anarchy for pure equilibria is* $\Theta(\sqrt{n})$ *for asymmetric but at most 5/2 for symmetric congestion games.*

Theorem 5 ([5]) *The price of anarchy for pure equilibria is at least* $\Omega(n^{p/(p+1)})$ *and at most* $O(n)$ *for asymmetric, but at most 5/2 for symmetric congestion games with polynomial latencies.*

Average Social Cost – Total Latency

Here, it is considered as social cost the sum of the players cost (divided by the number of the players)

$$\mathrm{SC}(A) = \textsc{Sum}(A) = \sum_{i \in N} c_i(A)$$

or the weighted sum of the players costs (total latency) for weighted games

$$\mathrm{SC}(A) = \mathrm{C}(A) = \sum_{i \in N} w_i c_i(A) .$$

The definition naturally extends for mixed strategies.

Theorem 6 ([2,4,5]) *The price of anarchy is* 5/2 *for asymmetric and* $(5n-2)/(2n+1)$ *for symmetric congestion games.*

Theorem 7 ([2,4]) *The Price of Anarchy for weighted congestion games is* $1+\phi \approx 2.618$.

Theorem 8 ([6]) *The Price of Anarchy is at most* 3/2 *for congestion games with splittable weights.*

Theorem 9 ([14,15]) *The Price of Anarchy for non-atomic congestion games is* 4/3.

Theorem 10 ([1,2,4]) *The Price of Anarchy for (weighted) congestion games is* $d^{\Theta(p)}$ *for polynomial latencies.*

Applications

The efficiency of large scale networks, in which selfish users interact, is highly affected due to the users' selfish behavior. The Price of Anarchy is a quantitative measure of the lack of coordination in such systems. It is a useful theoretical tool for the analysis and design of telecommunication and traffic networks, where selfish users compete on system's resources motivated by their atomic interests and are indifferent to the social welfare.

Cross References

► Best Response Algorithms for Selfish Routing
► Computing Pure Equilibria in the Game of Parallel Links
► Price of Anarchy for Machines Models

Recommended Reading

1. Aland, S., Dumrauf, D., Gairing, M., Monien, B., Schoppmann, F.: Exact price of anarchy for polynomial congestion games. In: 23rd Annual Symposium on Theoretical Aspects of Computer Science (STACS), pp. 218–229. Springer, Marseille (2006)
2. Awerbuch, B., Azar, Y., Epstein A.: Large the price of routing unsplittable flow. In: Proc. of the 37th Annual ACM Symposium on Theory of Computing (STOC), pp. 57–66. ACM, Baltimore (2005)
3. Awerbuch, B., Azar, Y., Richter, Y., Tsur, D.: Tradeoffs in worst-case equilibria. In: Approximation and Online Algorithms, 1st International Workshop (WAOA), pp. 41–52. Springer, Budapest (2003)
4. Christodoulou, G., Koutsoupias, E.: On the price of anarchy and stability of correlated equilibria of linear congestion games. In: Algorithms – ESA 2005, 13th Annual European Symposium, pp. 59–70. Springer, Palma de Mallorca (2005)
5. Christodoulou, G., Koutsoupias, E.: The price of anarchy of finite congestion games. In: Proc. of the 37th Annual ACM Symposium on Theory of Computing (STOC), pp. 67–73. ACM, Baltimore (2005)
6. Cominetti, R., Correa, J.R., Moses, N.E.S.: Network games with atomic players. In: Automata, Languages and Programming, 33rd International Colloquium (ICALP), pp. 525–536. Springer, Venice (2006)
7. Czumaj, A., Vöcking, B.: Tight bounds for worst-case equilibria. In: Proc. of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 413–420. ACM/SIAM, San Francisco (2002)
8. Koutsoupias, E., Mavronicolas, M., Spirakis, P.G.: Approximate equilibria and ball fusion. Theor. Comput. Syst. **36**, 683–693 (2003)
9. Koutsoupias, E., Papadimitriou, C.H.: Worst-case equilibria. In: Proc. of the 16th Annual Symposium on Theoretical Aspects of Computer Science (STACS), pp. 404–413. Springer, Trier (1999)
10. Mavronicolas, M., Spirakis, P.G.: The price of selfish routing. In: Proc. on 33rd Annual ACM Symposium on Theory of Computing (STOC), pp. 510–519. ACM, Heraklion (2001)
11. Monderer, D., Shapley, L.: Potential games. Games Econ. Behav. **14**, 124–143 (1996)
12. Papadimitriou, C.H.: Algorithms, games, and the internet. In: Proc. on 33rd Annual ACM Symposium on Theory of Computing (STOC), pp. 749–753. ACM, Heraklion (2001)
13. Rosenthal, R.W.: A class of games possessing pure-strategy Nash equilibria. Int. J. Game Theor. **2**, 65–67 (1973)
14. Roughgarden, T., Tardos, E.: How bad is selfish routing? J. ACM **49**, 236–259 (2002)
15. Roughgarden, T., Tardos, E.: Bounding the inefficiency of equilibria in nonatomic congestion games. Games Econ. Behav. **47**, 389–403 (2004)

Price of Anarchy for Machines Models

2002; Czumaj, Vöcking

Artur Czumaj[1], Berthold Vöcking[2]

[1] DIMAP and Computer Science, University of Warwick, Coventry, UK

[2] Department of Computer Science, RWTH Aachen University, Aachen, Germany

Keywords and Synonyms

Worst-case coordination ratio; Selfish routing

Problem Definition

Notations

This entry considers a selfish routing model formally introduced by Koutsoupias and Papadimitriou [11], in which the goal is to route the traffic on parallel links with linear latency functions. One can describe this model as a scheduling problem with m independent machines with speeds $s_1, \dots, s_m$ and n independent tasks with weights $w_1, \dots, w_n$. The goal is to allocate the tasks to the machines to minimize the maximum load of the links in the system.

It is assumed that all tasks are assigned by non-cooperative agents. The set of *pure strategies* for task i is the set $\{1, \dots, m\}$ and a *mixed strategy* is a distribution on this set.

Given a combination $(j_1, \dots, j_n) \in \{1, \dots, m\}^n$ of pure strategies, one for each task, the *cost* for task i is $\sum_{j_k = j_i} \frac{w_k}{s_{j_i}}$, which is the time needed for machine j_i chosen by task i to complete all tasks allocated to that machine. Similarly, for a combination of pure strategies $(j_1, \dots, j_n) \in \{1, \dots, m\}^n$, the *load* of machine j is defined as $\sum_{j_k = j} \frac{w_k}{s_j}$.

Given n tasks of length $w_1, \dots, w_n$ and m machines with the speeds $s_1, \dots, s_m$, let $\mathfrak{opt}$ denote the *social optimum*, that is, the minimum cost over all combinations of pure strategies:

$$\mathfrak{opt} = \min_{(j_1, \dots, j_n) \in \{1, \dots, m\}^n} \max_{1 \le j \le m} \sum_{i : j_i = j} \frac{w_i}{s_j} .$$

For example, if all machines have the same unit speed ($s_j = 1$ for every j, $1 \le j \le m$) and all tasks have the same unit weight ($w_i = 1$ for every i, $1 \le i \le n$), then the social optimum is $\lceil \frac{n}{m} \rceil$.

It is also easy to see that in any system

$$\mathfrak{opt} \ge \frac{\max_i w_i}{\max_j s_j} .$$

It is known that computing the social optimum is $\mathcal{NP}$-hard even for identical speeds (see [11]).

For mixed strategies, let p_i^j denote the probability that an agent i sends the entire traffic w_i to a machine j. Let ℓ_j denote the *expected load* on a machine j, that is,

$$\ell_j = \frac{1}{s_j} \cdot \sum_{i=1}^{n} w_i p_i^j .$$

For a task i, the *expected cost of task i on machine j* is equal to

$$c_i^j = \frac{w_i}{s_j} + \sum_{t \ne i} \frac{w_t p_t^j}{s_j} = \ell_j + (1 - p_i^j) \frac{w_i}{s_j} .$$

The expected cost c_i^j corresponds to the expected finish time of task i on machine j under the processor sharing scheduling policy. This is an appropriate cost model with respect to the underlying traffic routing application.

Definition 1 (Nash equilibrium) The probabilities $(p_i^j)_{1 \le i \le n, 1 \le j \le m}$ define a *Nash equilibrium* if and only if any task i will assign non-zero probabilities only to machines that minimize c_i^j, that is, $p_i^j > 0$ implies $c_i^j \le c_i^q$, for every q, $1 \le q \le m$.

As an example, in the system considered above in which all machines have the same unit speed and all weights are the same, the uniform probabilities $p_i^j = \frac{1}{m}$ for all $1 \le j \le m$ and $1 \le i \le n$ define a system in a Nash equilibrium.

The existence of a Nash equilibrium over mixed strategies for non-cooperative games was shown by Nash [13]. In fact, the routing game considered here admits an equilibrium even if all players are restricted to pure strategies, what has been shown by Fotakis et al. [7].

Fix an arbitrary Nash equilibrium, that is, fix the probabilities $(p_i^j)_{1 \le i \le n, 1 \le j \le m}$ that define a Nash equilibrium. Consider the randomized allocation strategies in which each task i is allocated to a single machine chosen independently at random according to the probabilities p_i^j, that is, task i is allocated to machine j with probability p_i^j. Let C_j, $1 \le j \le m$, be the random variable indicating the *load of machine j* in our random experiment. Observe that C_j is the weighted sum of independent 0–1 random variables J_i^j, $\mathbf{Pr}[J_i^j = 1] = p_i^j$, such that

$$C_j = \frac{1}{s_j} \sum_{i=1}^{n} w_i \cdot J_i^j .$$

Let $\mathfrak{c}$ denote the *maximum expected load* over all machines, that is,

$$\mathfrak{c} = \max_{1 \le j \le m} \ell_j .$$

Notice that $\mathbf{E}[C_j] = \ell_j$, and therefore $\mathfrak{c} = \max_{1 \le j \le m} \mathbf{E}[C_j]$.

Finally, let the *social cost* C be defined as the expected maximum load (instead of maximum expected load), that is,

$$\mathsf{C} = \mathbf{E}[\max_{1 \le j \le m} C_j] .$$

Observe that $c \le C$ and possibly $c \ll C$. The goal is to estimate the *price of anarchy* (also called the *worst-case coordination ratio*) which is the worst-case ratio

$$R = \max \frac{\mathsf{C}}{\mathfrak{opt}} ,$$

where the maximum is over all Nash equilibria.

Key Results

Early Work

The study of the price of anarchy has been initiated by Koutsoupias and Papadimitriou [11], who showed also some very basic results for this model. For example, they proved that for two identical machines the price of anarchy is exactly $\frac{3}{2}$, and for two machines (with possibly different speeds) the price of anarchy is at least $\phi = (1 + \sqrt{5})/2$. Koutsoupias and Papadimitriou showed also that for m identical machines the price of anarchy is $\Omega(\log m/(\log\log m))$ and it is at most $\mathcal{O}(\sqrt{m \ln m})$, and for m arbitrary machines the price of anarchy is $\mathcal{O}(\sqrt{s_1/s_m \sum_{j=1}^{m} s_j/s_m} \sqrt{\log m})$, where $s_1 \geq s_2 \geq \cdots \geq s_m$ [11].

Koutsoupias and Papadimitriou [11] conjectured also that the price of anarchy for m *identical machines* is $\Theta(\log m/(\log\log m))$. In the quest to resolve this conjecture, Mavronicolas and Spirakis [12] considered the problem in the so-called *fully-mixed model*, which is a special class of Nash equilibria in which all p_i^j are strictly positive. In this model, Mavronicolas and Spirakis [12] showed that for m identical machines in the fully-mixed Nash equilibrium the price of anarchy is $\Theta(\log m(\log\log m))$. Similarly, they proved also that for m (not necessarily identical) machines and n identical weights in the fully-mixed Nash equilibrium, if $m \leq n$, then the price of anarchy is $\Theta(\log n/(\log\log n))$.

The motivation behind studying fully-mixed equilibria is the so-called fully-mixed Nash equilibrium conjecture stating that these equilibria maximize the price of anarchy because they maximize the randomization. The conjecture seems to be quite appealing as a fully-mixed equilibrium can be computed in polynomial time, which led to numerous studies of this kind of equilibria with the hope to obtain efficient algorithms for computing or approximating the price of anarchy with respect to mixed equilibria. However, Fischer and Vöcking [6] disproved the fully-mixed Nash equilibrium conjecture and showed that there is a mixed Nash equilibrium whose expected cost is larger than the expected cost of the fully-mixed Nash equilibrium by a factor of $\Omega(\log m/(\log\log m))$. Furthermore, they presented polynomial time algorithms for approximating the price of anarchy for mixed equilibria on identical machines up to a constant factor.

Tight Bounds for the Price of Anarchy

Czumaj and Vöcking [4] entirely resolved the conjecture of Koutsoupias and Papadimitriou [11] and gave an exact description of the price of anarchy as a function of the number of machines and the ratio of the speed of the fastest machine over the speed of the slowest machine.[1]

Theorem 1 [4] (Upper Bound) *The price of anarchy for m machines is bounded from above by*

$$\mathcal{O}\left(\min\left\{\frac{\log m}{\log\log\log m}, \frac{\log m}{\log\left(\frac{\log m}{\log(s_1/s_m)}\right)}\right\}\right),$$

where it is assumed that the speeds satisfy $s_1 \geq \cdots \geq s_m$.

In particular, the price of anarchy for m machines is $\mathcal{O}(\frac{\log m}{\log\log\log m})$.

The theorem follows directly from the following two results [4]: that the maximum expected load $\mathfrak{c}$ satisfies

$$\begin{aligned}\mathfrak{c} &= \mathfrak{opt} \cdot \Gamma^{(-1)}(m)\\ &= \mathfrak{opt} \cdot \mathcal{O}\left(\min\left\{\frac{\log m}{\log\log m}, \log\left(\frac{s_1}{s_m}\right)\right\}\right)\end{aligned}$$

and that the social cost C satisfies

$$\mathsf{C} = \mathfrak{opt} \cdot \mathcal{O}\left(\frac{\log m}{\log\left(\frac{\mathfrak{opt}\cdot\log m}{\mathfrak{c}}\right)} + 1\right).$$

If one applied these results to systems in which all agents follow only *pure strategies*, then since then $\ell_j = C_j$ for every j, it holds that $\mathsf{C} = \mathfrak{c}$. This leads to the following result.

Corollary 2 [4] *For pure strategies the price of anarchy for m machines is upper bounded by*

$$\mathcal{O}\left(\min\left\{\frac{\log m}{\log\log m}, \log\left(\frac{s_1}{s_m}\right)\right\}\right),$$

where it is assumed that the speeds satisfy $s_1 \geq \cdots \geq s_m$.

Theorem 4 below proves that this corollary gives an asymptotically tight bound for the price of anarchy for pure
strategies.

By Theorem 1, in the special case when all *machines are identical*, the price of anarchy is $\mathcal{O}(\log m/(\log\log m))$;

[1]To simplify the notation, for any real $x \geq 0$, let $\log x$ denote $\log x = \max\{\log_2 x, 1\}$. Also, following standard convention, $\Gamma(N)$ is used to denote the *Gamma (factorial) function*, which for any natural N is defined by $\Gamma(N+1) = N!$ and for an arbitrary real $x > 0$ is $\Gamma(x) = \int_0^\infty t^{x-1} e^{-t}\,dt$. For the inverse of the Gamma function, $\Gamma^{(-1)}(N)$, it is known that $\Gamma^{(-1)}(N) = x$ such that $\lfloor x \rfloor! \leq N - 1 \leq \lceil x \rceil!$. It is well known that $\Gamma^{(-1)}(N) = (\log N)/(\log\log N)(1 + o(1))$.

this result has been also obtained independently by Koutsoupias et al. [10]. However, in this special case one can get a stronger bound that is tight up to an additive constant.

Theorem 3 [4] *For m identical machines the price of anarchy is at most*

$$\Gamma^{(-1)}(m) + \Theta(1) = \frac{\log m}{\log\log m} \cdot (1 + o(1)) \,.$$

One can obtain a lower bound for the price of anarchy for *m identical machines* by considering the system in which $p_i^j = \frac{1}{m}$ for every i, j. The Result of Gonnet [9] implies that then the price of anarchy is $\Gamma^{(-1)}(m) - \frac{3}{2} + o(1)$, which implies that Theorem 3 is tight up to an additive constant.

The next theorem shows that the upper bound in Theorem 1 is asymptotically tight.

Theorem 4 [4] **(Lower bound)** *The price of anarchy for m machines is lower bounded by*

$$\Omega\left(\min\left\{\frac{\log m}{\log\log\log m}, \frac{\log m}{\log\left(\frac{\log m}{\log(s_1/s_m)}\right)}\right\}\right) \,.$$

In particular, the price of anarchy for m machines is $\Omega(\log m/(\log\log\log m))$.

In fact, it can be shown [4] (analogously to the upper bound) that for every positive integer m, positive real r, and $S \geq 1$, there exists a set of m machines with $\frac{s_1}{s_m} = S$ being in a Nash equilibrium and satisfying $\mathfrak{opt} = r$,

$$\mathfrak{c} = \mathfrak{opt} \cdot \Omega\left(\min\left\{\frac{\log m}{\log\log m}, \log\left(\frac{s_1}{s_m}\right)\right\}\right) \,,$$

and

$$\mathsf{C} = \mathfrak{opt} \cdot \Omega\left(\frac{\log m}{\log\left(\frac{\mathfrak{opt}\cdot\log m}{c}\right)}\right) \,.$$

Applications

The model discussed here has been extended in the literature in numerous ways, in particular in [1,5,8]; see also survey presentations in [3,14].

Open Problems

An interesting attempt that adds an algorithmic or constructive element to the analysis of the price of anarchy is made in [2]. The idea behind "coordination mechanisms" is not to study the price of anarchy for a fixed system, but to design the system in such a way that the increase in cost or the loss in performance due to selfish behavior is as small as possible. This is a promising direction of research that might result in practical guidelines of how to build a distributed system that does not suffer from selfish behavior but might even exploit the selfishness of the agents.

Cross References

► Computing Pure Equilibria in the Game of Parallel Links
► Price of Anarchy

Recommended Reading

1. Awerbuch, B., Azar, Y., Richter, Y., Tsur, D.: Tradeoffs in worst-case equlibria. Theor. Comput. Sci. **361**, 200–209 (2006)
2. Christodoulou, G., Koutsoupias, E., Nanavati, A.: Coordination mechanisms. In: Proceedings of the 31st International Colloquium on Automata, Languages and Programming (ICALP), pp. 345–357 (2004)
3. Czumaj, A.: Selfish routing on the Internet. In: Leung, J. (ed.) Handbook of Scheduling: Algorithms, Models, and Performance Analysis. CRC Press, Boca Raton, FL, USA (2004)
4. Czumaj, A., Vöcking, B.: Tight bounds for worst-case equilibria. ACM Trans. Algorithms **3**(1) (2007)
5. Czumaj, A., Krysta, P., Vöcking, B.: Selfish traffic allocation for server farms. In: Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC), pp. 287–296 (2002)
6. Fischer, S., Vöcking, B.: On the structure and complexity of worst-case equilibria. Theor. Comput. Sci. **378**(2), 165–174 (2007)
7. Fotakis, D., Kontogiannis, S., Koutsoupias, E., Mavronicolas, M., Spirakis, P.: The structure and complexity of Nash equilibria for a selfish routing game. In Proceedings of the 29th International Colloquium on Automata, Languages and Programming (ICALP), pp. 123–134, (2002)
8. Gairing, M., Lücking, T., Mavronicolas, M., Monien, B.: The price of anarchy for polynomial social cost. Theor. Comput. Sci. **369**(1-3), 116–135 (2006)
9. Gonnet, G.: Expected length of the longest probe sequence in hash code searching. J. Assoc. Comput. Mach. **28**(2), 289–304 (1981)
10. Koutsoupias, E., Mavronicolas, M., Spirakis, P.: Approximate equilibria and ball fusion. Theor. Comput. Syst. **36**(6), 683–693 (2003)
11. Koutsoupias, E., Papadimitriou, C.H.: Worst-case equilibria. In: Proceedings of the 16th Annual Symposium on Theoretical Aspects of Computer Science (STACS), pp. 404–413 (1999)
12. Mavronicolas, M., Spirakis, P.: The price of selfish routing. In: Proceedings of the 33rd Annual ACM Symposium on Theory of Computing (STOC), pp. 510–519 (2001)
13. Nash Jr., J.F.: Non-cooperative games. Ann. Math. **54**(2), 286–295 (1951)
14. Vöcking, B.: Selfish load balancing. In: Nisan, N., Roughgarden, T., Tardos, É., Vazirani, V. (eds.) Algorithmic Game Theory. Cambridge University Press, New York, NY, USA (2007)

Probabilistic Data Forwarding in Wireless Sensor Networks

2004; Chatzigiannakis, Dimitriou, Nikoletseas, Spirakis

SOTIRIS NIKOLETSEAS
Computer Engineering and Informatics, Department and CTI, University of Patras, Patras, Greece

Keywords and Synonyms

Data propagation; Routing

Problem Definition

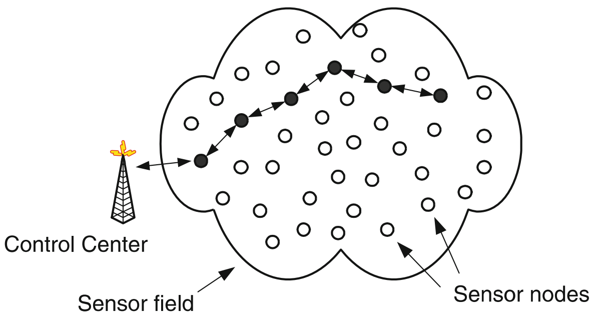
An important problem in wireless sensor networks is that of *local detection and propagation*, i. e. the local sensing of a crucial event and the energy and time efficient propagation of data reporting its realization to a control center (for a graphical presentation, see Fig. 1). This center (called the *"sink"*) could be some human authorities responsible of taking action upon the realization of the crucial event. More formally:

Definition 1 Assume that a single sensor, E, senses the realization of a *local event* $\mathcal{E}$. Then the *propagation problem* is the following: "How can sensor P, via cooperation with the rest of the sensors in the network, efficiently propagate information reporting the realization of the event to the sink S?"

Note that this problem is in fact closely related to the more general problem of data propagation in sensor networks.

Wireless Sensor Networks

Recent dramatic developments in micro-electro-mechanical systems (MEMS), wireless communications and digital electronics have led to the development of small in size, low-power, low-cost sensor devices. Such extremely small (soon in the cubic millimetre scale) devices integrate sensing, data processing and wireless communication capabilities. Examining each such device individually might appear to have small utility, however the effective *distributed self-organization* of large numbers of such devices into an ad-hoc network may lead to the efficient accomplishment of large sensing tasks. Their wide range of applications is based on the use of various sensor types (i. e. thermal, visual, seismic, acoustic, radar, magnetic, etc.) to monitor a wide variety of conditions (e. g. temperature, object presence and movement, humidity, pressure, noise levels etc.). For a survey on wireless sensor networks see [1] and also [6,9].



Probabilistic Data Forwarding in Wireless Sensor Networks, Figure 1
A sensor network

A Simple Model

Sensor networks are comprised of a vast number of ultra-small homogeneous sensors, which are called *"grain" particles*. Each grain particle is a fully-autonomous computing and communication device, characterized mainly by its available power supply (battery) and the energy cost of computation and transmission of data. Such particles (in the model here) do not move. Each particle is equipped with a set of monitors (sensors) for light, pressure, humidity, temperature etc. and has a *broadcast* (digital radio) *beacon mode*.

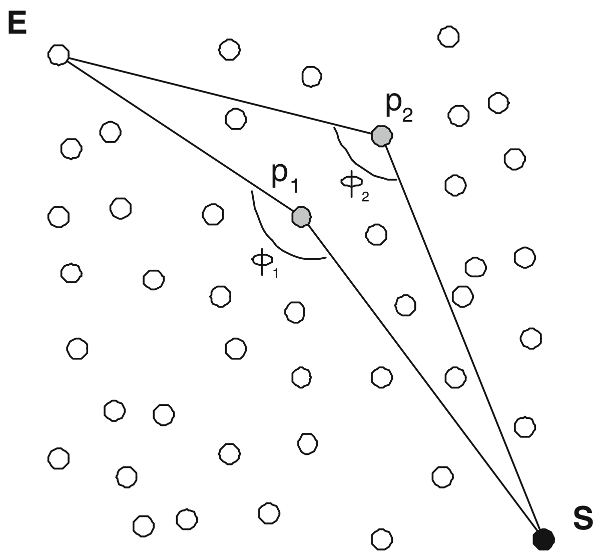
It is assumed that grain particles are *randomly deployed* in a given area of interest. Such a placement may occur e. g. when throwing sensors from an airplane over an area. A special case is considered, when the network being a lattice (or grid) deployment of sensors. This grid placement of grain particles is motivated by certain applications, where it is possible to have a pre-deployed sensor network, where sensors are put (possibly by a human or a robot) in a way that they form a *2-dimensional lattice*.

It is assumed that each particle has the following abilities: (i) It can estimate the direction of a received transmission (e. g. via the technology of direction-sensing antennae). (ii) It can estimate the distance from a nearby particle that did the transmission (e. g. via estimation of the attenuation of the received signal). (iii) It knows the direction towards the sink S. This can be implemented during a set-up phase, where the (powerful) sink broadcasts the information about itself to all particles. (iv) All particles have a common co-ordinates system. Notice that GPS information is not assumed. Also, there is no need to know the global structure of the network.

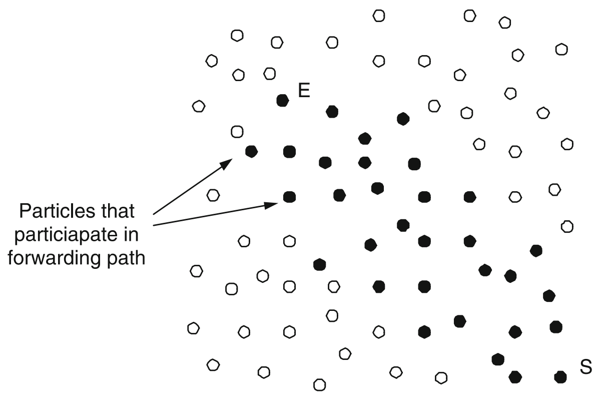
Key Results

The Basic Idea

For the above problem [3] proposes a protocol which tries to minimize energy consumption by *probabilistically fa-*



Probabilistic Data Forwarding in Wireless Sensor Networks, Figure 2
Angle φ and proximity to the optimal line



Probabilistic Data Forwarding in Wireless Sensor Networks, Figure 3
Thin zone of particles

voring certain paths of local data transmissions towards the sink. Thus this protocol is called PFR (Probabilistic Forwarding Protocol). Its basic idea is to *avoid flooding* by favoring (in a probabilistic manner) data propagation along sensors which lie "close" to the (optimal) transmission line, *ES*, that connects the sensor node detecting the event, *E*, and the sink, *S*. This is implemented by locally calculating the angle $\phi = (\widehat{EPS})$, whose corner point *P* is the sensor currently running the local protocol, having received a transmission from a nearby sensor, previously possessing the event information (see Fig. 2). If φ is equal or greater to a predetermined threshold, then *p* will transmit (and thus propagate the information further). Else, it decides whether to transmit with probability equal to $\frac{\phi}{\pi}$. Because of the probabilistic nature of data propagation decisions and to prevent the propagation process from early failing, the protocol initially uses (for a short time period which is evaluated) a flooding mechanism that leads to a sufficiently large *"front"* of sensors possessing the data under propagation. When such a "front" is created, probabilistic Forwarding is performed.

The PFR Protocol

The protocol evolves in two phases:

Phase 1: The "Front" Creation Phase Initially the protocol builds (by using a limited, in terms of rounds, flooding) a sufficiently large "front" of particles, to guarantee the survivability of the data propagation process. During this phase, each particle having received the data to be propagated, deterministically forwards them towards the sink.

Phase 2: The Probabilistic Forwarding Phase Each particle *P* possessing the information under propagation (called $info(\mathcal{E})$ hereafter), calculates an angle φ by calling the subprotocol "φ-calculation" (see description below) and broadcasts $info(\mathcal{E})$ to all its neighbors with probability $\mathbb{P}_{fwd}$ (or it does not propagate any data with probability $1 - \mathbb{P}_{fwd}$) as follows:

$$\mathbb{P}_{fwd} = \begin{cases} 1 & \text{if } \phi \geq \phi_{\text{threshold}} \\ \frac{\phi}{\pi} & \text{otherwise} \end{cases}$$

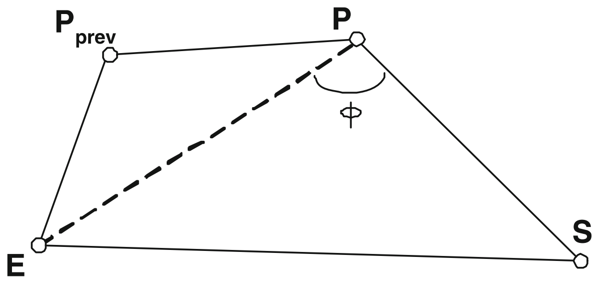
where φ is the $(\widehat{EPS})$ angle and $\phi_{\text{threshold}} = 134°$ (the selection reasons of this value are discussed in [3]).

If the density of particles is appropriately large, then for a line *ES* there is (with high probability) a sequence of points "closely surrounding *ES*" whose angles φ are larger than $\phi_{\text{threshold}}$ and so that successive points are within transmission range. All such points broadcast and thus essentially they follow the line *ES* (see Fig. 3).

The φ-calculation Subprotocol (see Fig. 4)

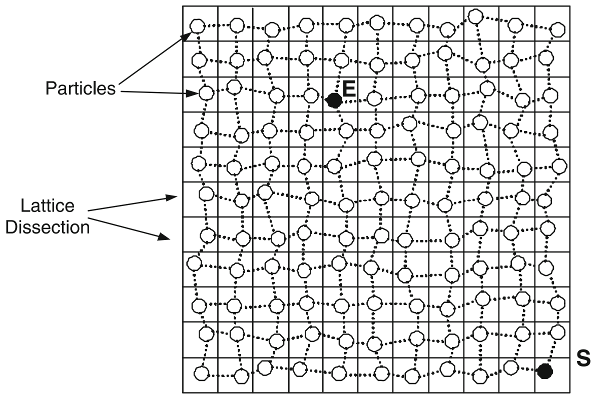
Let P_{prev} the particle that transmitted $info(E)$ to *P*.

1. When P_{prev} broadcasts $info(E)$, it also attaches the info $|EP_{\text{prev}}|$ and the direction $\overrightarrow{P_{\text{prev}}E}$.
2. *P* estimates the direction and length of line segment $P_{\text{prev}}P$, as described in the model.
3. *P* now computes angle $(\widehat{EP_{\text{prev}}P})$, and computes $|EP|$ and the direction of $\overrightarrow{PE}$ (this will be used in further transmission from *P*).



Probabilistic Data Forwarding in Wireless Sensor Networks, Figure 4
Angle φ calculation example

4. P also computes angle $(\widehat{P_{\text{prev}}PE})$ and by subtracting it from $(\widehat{P_{\text{prev}}PS})$ it finds φ.



Probabilistic Data Forwarding in Wireless Sensor Networks, Figure 5
A lattice dissection G

Performance Properties of PFR

Any protocol Π solving the data propagation problem must satisfy the following three properties: **a) Correctness**. Π must guarantee that data arrives to the position S, given that the whole network exists and is operational. **b) Robustness**. Π must guarantee that data arrives at enough points in a small interval around S, in cases where part of the network has become inoperative. **c) Efficiency**. If Π activates k particles during its operation then Π should have a small ratio of the number of activated over the total number of particles $r = \frac{k}{N}$. Thus r is an energy efficiency measure of Π. It is shown that this is indeed the case for PFR.

Consider a partition of the network area into small squares of a fictitious grid G (see Fig. 5). When particle density is high enough, occupancy arguments guarantee that with very high probability (tending to 1) all squares get particles. All the analysis is conditioned on this event, call it F, of at least one particle in each square. Below only sketches of proofs are provided (full proofs can be found in [3]).

The Correctness of PFR

Consider any square Σ intersecting the ES line. By the occupancy argument above, there is w.h.p. a particle in this square. Clearly, the worst case is when the particle is located in one of the corners of Σ (since the two corners located most far away from the ES line have the smallest φ-angle among all positions in Σ). By geometric calculations, [3] proves that the angle φ of this particle is $\phi > 134°$. But the initial square (i. e. that containing E) always broadcasts and any intermediate intersecting square will be notified (by induction) and thus broadcast because of the argument above. Thus the sink will be reached if the whole network is operational:

Lemma 1 ([3]) *PFR succeeds with probability* 1 *given the event F.*

The Energy Efficiency of PFR

Consider a "lattice-shaped" network like the one in Fig. 5 (all results will hold for any random deployment "in the limit"). The analysis of the energy efficiency considers particles that are active but are as far as possible from ES. [3] estimates an upper bound on the number of particles in an $n \times n$ (i. e. $N = n \times n$) lattice. If k is this number then $r = \frac{k}{n^2}$ $(0 < r \leq 1)$ is the "energy efficiency ratio" of PFR. More specifically, in [3] the authors prove the (very satisfactory) result below. They consider the area around the ES line, whose particles participate in the propagation process. The number of active particles is thus, roughly speaking, captured by the size of this area, which in turn is equal to $|ES|$ times the maximum distance from $|ES|$. This maximum distance is clearly a random variable. To calculate the expectation and variance of this variable, the authors in [3] basically "upper bound" the stochastic process of the distance from ES by a random walk on the line, and subsequently "upper bound" this random walk by a well-known stochastic process (i. e. the "discouraged arrivals" birth and death Markovian process. Thus they prove:

Theorem 2 ([3]) *The energy efficiency of the PFR protocol is* $\Theta\left(\left(\frac{n_0}{n}\right)^2\right)$ *where* $n_0 = |ES|$ *and* $n = \sqrt{N}$, *where N is the number of particles in the network. For* $n_0 = |ES| = o(n)$, *this is o(1).*

The Robustness of PFR

Consider particles "very near" to the *ES* line. Clearly, such particles have large φ-angles (i. e. $\phi > 134°$). Thus, even in the case that some of these particles are not operating, the probability that none of those operating transmits (during phase 2) is very small. Thus:

Lemma 3 ([3]) *PFR manages to propagate the crucial data across lines parallel to ES, and of constant distance, with* fixed *nonzero probability (not depending on n, $|ES|$).*

Applications

Sensor networks can be used for continuous sensing, event detection, location sensing as well as micro-sensing. Hence, sensor networks have several important applications, including (a) security (like biological and chemical attack detection), (b) environmental applications (such as fire detection, flood detection, precision agriculture), (c) health applications (like telemonitoring of human physiological data) and (d) home applications (e. g. smart environments and home automation). Also, sensor networks can be combined with other wireless networks (like mobile) or fixed topology infrastructures (like the Internet) to provide transparent wireless extensions in global computing scenaria.

Open Problems

It would be interesting to come up with formal models for sensor networks, especially with respect to energy aspects; in this respect, [10] models energy dissipation using stochastic methods. Also, it is important to investigate fundamental trade-offs, such as those between energy and time. Furthermore, the presence of mobility and/or multiple sinks (highly motivated by applications) creates new challenges (see e. g. [2,11]). Finally, heterogeneity aspects (e. g. having sensors of various types and/or combinations of sensor networks with other types of networks like p2p, mobile and the Internet) are very important; in this respect see e. g. [5,13].

Experimental Results

An implementation of the PFR protocol along with a detailed comparative evaluation (using simulation) with greedy forwarding protocols can be found in [4]; with clustering protocols (like LEACH, [7]) in [12]; with tree maintenance approaches (like Directed Diffusion, [8]) in [5]. Several performance measures are evaluated, like the success rate, the latency and the energy dissipation. The simulations mainly suggest that PFR behaves best in sparse networks of high dynamics.

Cross References

- ► Communication in Ad Hoc Mobile Networks Using Random Walks
- ► Obstacle Avoidance Algorithms in Wireless Sensor Networks
- ► Randomized Energy Balance Algorithms in Sensor Networks

Recommended Reading

1. Akyildiz, I.F., Su, W., Sankarasubramaniam, Y., Cayirci, E.: Wireless sensor networks: a survey. J. Comput. Netw. **38**, 393–422 (2002)
2. Chatzigiannakis, I., Kinalis, A., Nikoletseas, S.: Sink Mobility Protocols for Data Collection in Wireless Sensor Networks . In: Proc. of the 4th ACM/IEEE International Workshop on Mobility Management and Wireless Access Protocols (MobiWac), ACM Press, pp. 52–59 (2006)
3. Chatzigiannakis, I., Dimitriou, T., Nikoletseas, S., Spirakis, P.: A Probabilistic Algorithm for Efficient and Robust Data Propagation in Smart Dust Networks. In: Proc. 5th European Wireless Conference on Mobile and Wireless Systems (EW 2004), pp. 344–350 (2004). Also in: Ad-Hoc Netw J **4**(5), 621–635 (2006)
4. Chatzigiannakis, I., Dimitriou, T., Mavronicolas, M., Nikoletseas, S., Spirakis, P.: A Comparative Study of Protocols for Efficient Data Propagation in Smart Dust Networks. In: Proc. 9th European Symposium on Parallel Processing (EuroPar), Distinguished Paper. Lecture Notes in Computer Science, vol. 2790, pp. 1003–1016. Springer (2003) Also in the Parallel Processing Letters (PPL) Journal, Volume 13, Number 4, pp. 615–627 (2003)
5. Chatzigiannakis, I., Kinalis, A., Nikoletseas, S.: An Adaptive Power Conservation Scheme for Heterogeneous Wireless Sensors. In: Proc. 17th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA 2005), ACM Press, pp. 96–105 (2005). Also in: Theory Comput Syst (TOCS) J **42**(1), 42–72 (2008)
6. Estrin, D., Govindan, R., Heidemann, J., Kumar, S.: Next Century Challenges: Scalable Coordination in Sensor Networks. In: Proc. 5th ACM/IEEE International Conference on Mobile Computing, MOBICOM'1999
7. Heinzelman, W.R., Chandrakasan, A., Balakrishnan, H.: Energy-Efficient Communication Protocol for Wireless Microsensor Networks. In: Proc. 33rd Hawaii International Conference on System Sciences, HICSS'2000
8. Intanagonwiwat, C., Govindan, R., Estrin, D.: Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks. In: Proc. 6th ACM/IEEE International Conference on Mobile Computing, MOBICOM'2000
9. Kahn, J.M., Katz, R.H., Pister, K.S.J.: Next Century Challenges: Mobile Networking for Smart Dust. In: Proc. 5th ACM/IEEE International Conference on Mobile Computing, pp. 271–278, Sept. 1999

10. Leone, P., Rolim, J., Nikoletseas, S.: An Adaptive Blind Algorithm for Energy Balanced Data Propagation in Wireless Sensor Networks. In: Proc. of the IEEE International Conference on Distributed Computing in Sensor Networks (DCOSS). Lecture Notes in Computer Science (LNCS), vol. 3267, pp. 35–48. Springer (2005)
11. Luo, J., Hubaux, J.-P.: Joint Mobility and Routing for Lifetime Elongation in Wireless Networks. In: Proc. 24th INFOCOM (2005)
12. Nikoletseas, S., Chatzigiannakis, I., Antoniou, A., Efthymiou, C., Kinalis, A., Mylonas, G.: Energy Efficient Protocols for Sensing Multiple Events in Smart Dust Networks. In: Proc. 37th Annual ACM/IEEE Simulation Symposium (ANSS'04), pp. 15–24, IEEE Computer Society Press (2004)
13. Triantafillou, P., Ntarmos, N., Nikoletseas, S., Spirakis, P.: NanoPeer Networks and P2P Worlds. In:Proc. 3rd IEEE International Conference on Peer-to-Peer Computing (P2P 2003), pp. 40–46, Sept. 2003

Q

Quantization of Markov Chains

2004; Szegedy

PETER RICHTER, MARIO SZEGEDY
Department of Computer Science, Rutgers,
The State University of New Jersey, Piscataway, NJ, USA

Keywords and Synonyms

Quantum walks

Problem Definition

Spatial Search and Walk Processes

Spatial search by *quantum walk* is database search with the additional constraint that one must move through the search space via a *quantum walk* that obeys some locality structure (grid, hypercube, etc.). Quantum walks are analogues of classical random walks on graphs. The complexity of spatial search by quantum walk is essentially determined by the *quantum hitting time* [9] of the walk.

Let S with $|S| = N$ be a finite set of *states*, and let $P = (p_{x,y})_{x,y \in S}$ be the *transition probability matrix* of a *Markov chain* on S, also denoted by P. Assume that a subset $M \subseteq S$ of states are *marked*. The goal is either to find a marked state, given that $M \neq \emptyset$ (*search version*), or to determine whether M is nonempty (*decision version*). If the possible $x \to y$ moves (i. e., those with $p_{x,y} \neq 0$) form the edges of a (directed) graph G, it is said that the walk has *locality structure* G.

INPUT: Markov chain P on set S, marked subset $M \subseteq S$.
OUTPUT: A marked state with probability 0.1 iff one exists (search version), or a Boolean return value with one-sided error detecting $M \neq \emptyset$ with probability 0.1 (decision version).

If P is *irreducible* (i. e., if its underlying digraph is strongly connected), a marked state can be found with high probability in finite time by simulating a *classical* random walk using the coefficients of P. In the *quantum* case, this random walk process may be replaced by a *quantum walk* using the coefficients of P (in particular, respecting locality). The fundamental question is whether the quantum walk process finds a marked state *faster* than the classical random walk process.

The Quantum Walk Algorithm

Quantizing P is not so straightforward, since stochastic matrices have no immediate unitary equivalents. It turns out that one must either abandon the discrete-time nature of the walk [7] or define the walk operator on a space other than $\mathbf{C}^S$. Here the second route is taken, with notation as in [18]. On $\mathbf{C}^{S \times S}$, define the unitary $W_P := R_1 R_2$, where $R_1 = \sum_{x \in S}(2|p_x\rangle\langle p_x| - I) \otimes |x\rangle\langle x|$, $R_2 = \sum_{x \in S} |x\rangle\langle x| \otimes (2|p_x\rangle\langle p_x| - I)$, and $|p_x\rangle := \sum_{y \in S} \sqrt{p_{y,x}}|y\rangle$. W_P is the *quantization* of P, or the *discrete-time quantum walk operator* arising from P. One can "check" whether or not the current state is marked by applying the operator $O_M = \sum_{x \notin M} |x\rangle\langle x| - \sum_{x \in M} |x\rangle\langle x|$. Denote the cost of constructing W_P (in the units of the resource of interest) by U (*update cost*), the cost of constructing O_M by C (*checking cost*), and the cost of preparing the initial state, ϕ_0, by S (*setup cost*). Every time an operator is used, its cost is incurred. This abstraction, implicit in [2] and made explicit in [13], allows W_P and O_M to be treated as black-box operators and provides a convenient way to capture *time complexity* or, in the quantum query model, *query complexity*. The spatial search algorithm by quantum walk is described by:

ALGORITHM: A quantum circuit $X = X_m X_{m-1} \ldots X_1$, with "wires" (typically two) that carry $\mathbf{C}^S$, and control bits. Each X_i is either a W_P gate or an O_M gate, or a controlled version of one of these. X is applied to the initial state ϕ_0. The cost of the sequence is the sum of the costs of the individual operators. The *observation probability* is the probability that after measuring the final state, ϕ_m, in the standard basis, one of the wires outputs an element of M. If the observation probability is q, one must repeat the procedure $1/\sqrt{q}$ times using amplitude amplification (search version). In the decision version one can distinguish between M and M' if $|X\phi_0 - X'\phi_0| \geq 0.1$, where X arises from O_M and X' from $O_{M'}$.

Key Results

Earlier Results

Spatial search blends Grover's search algorithm [8], which finds a marked element in a database of size N in $\sqrt{N/|M|}$ steps, with quantum walks.

Quantum walks were first introduced by David Meyer and John Watrous to study quantum cellular automata and quantum log-space, respectively. Discrete-time quantum walks were investigated for their own sake by Nayak et al. [3,15] and Aharonov et al. [1] on the infinite line and the N-cycle, respectively. The central issues in the early development of quantum walks included definition of the walk operator, notions of mixing and hitting times, and speedups achievable compared to the classical setting. Exponential quantum speedup of the hitting time between antipodes of the hypercube was shown by Kempe [9], and Childs et al. [6] presented the first oracle problem solvable exponentially faster by a quantum walk based algorithm than by any (not necessarily walk-based) classical algorithm.

The first systematic studies of quantum hitting time on the hypercube and the d-dimensional torus were conducted by Shenvi et al. [17] and Ambainis et al. [4]. Improving upon the Grover search based spatial search algorithm of Aaronson and Ambainis, Ambainis et al. [4] showed that the d-dimensional torus (with $N = n^d$ nodes) can be searched by quantum walk with cost of order $S + \sqrt{N}(U + C)$ and observation probability $\Omega(1/\log N)$ for $d \geq 3$, and with cost of order $S + \sqrt{N \log N}(U + C)$ and observation probability $\Omega(1)$ for $d = 2$. The key difference between these results and those of [6,9] is that the walk is required to start from the uniform state, not from one which is somehow "related" to the state one wishes to hit. Only in the latter case is it possible to achieve an exponential speedup.

The first result that used a quantum walk to solve a natural algorithmic problem, the so-called *element distinctness problem*, was due to Ambainis [2]. Ambainis' algorithm uses a walk W on the *Johnson graph* $J(r, m)$ whose vertices are the r-size subsets of a universe of size m, with two subsets connected iff their symmetric difference has size two. The relevance of this graph follows from a nontrivial algorithmic idea whereby the three different costs (S, U, and C) are balanced in a novel way. In contrast, Grover's algorithm – though it inspired Ambainis' result – has no such option: its setup and update costs are zero in the query model.

Ambainis' main mathematical observation about the walk W on the Johnson graph is that $W^{\sqrt{r}} O_M$ behaves in much the same way as the Grover iteration DO_M, where D is the Grover diffusion operator. Recall that Grover's algorithm applies DO_M repeatedly, sending the uniform starting state ϕ_0 to the state $\phi_{good} = \sum_{x \in M} \sqrt{1/|M|}|x\rangle$ after $t = O(1/\alpha)$ iterations, where $\alpha := 2 \sin^{-1} \langle \phi_{\text{good}} | \phi_0 \rangle$ is the effective "rotation angle".

What do $W^{\sqrt{r}}$ and D have in common? Ambainis showed that the nontrivial eigenvalues of the matrix $W^{\sqrt{r}}$ in the (finite dimensional) subspace containing the orbit of ϕ_0 are separated away from 1 by a constant ε. Thus, $W^{\sqrt{r}}$ serves as a very good approximate reflection about the axis ϕ_0 – as good as Grover's in this application. This allows one to conclude the following: there exists a $t = O(1/\alpha)$ for which the overlap $\langle \phi_{\text{good}} | (W^{\sqrt{r}} O_M)^t | \phi_0 \rangle = \Omega(1)$, so the output is likely in M.

Theorem 1 ([2]) *Let P be the random walk on the Johnson graph $J(r, m)$ with $r = o(m)$. Let M be either the empty set or the set of all r-size subsets containing a fixed subset $x_1, \ldots, x_k$ for constant $k \leq r$. Then there is a quantum algorithm that solves the hitting problem (search version) with cost of order $S + t(\sqrt{r} \cdot U + C)$, where $t = (\frac{m}{r})^{k/2}$. If the costs are $S = r$, $U = O(1)$, and $C = 0$, then the total cost has optimum $O(m^{k/(k+1)})$ at $r = O(m^{k/(k+1)})$.*

General Markov Chains

In [18], Szegedy investigates the hitting time of quantum walks arising from general Markov chains. His definitions (walk operator, hitting time) are abstracted directly from [2] and are consistent with prior literature, although slightly different in presentation.

For a Markov chain P, the (classical) *average hitting time* with respect to M can be expressed in terms of the *leaking walk matrix* P_M, which is obtained from P by deleting all rows and columns indexed by states of M. Let $h(x, M)$ denote the expected time to reach M from x and let $v_1, \ldots, v_{N-|M|}, \lambda_1, \ldots, \lambda_{N-|M|}$ be the (normalized) eigenvectors and associated eigenvalues of P_M. Let $d : S \to \mathbf{R}^+$ be a starting distribution and d' its restriction to $S \setminus M$. Then $h := \sum_{x \in S} d(x) h(x, M) = \sum_{k=1}^{N-|M|} \frac{|(v_k, d')|^2}{1-\lambda_k}$. Although the leaking walk matrix P_M is not stochastic, one can consider the *absorbing walk* matrix $P' = \left[\begin{smallmatrix} P_M & 0 \\ P'' & I \end{smallmatrix}\right]$, where P'' is the matrix obtained from P by deleting columns indexed by M and rows indexed by $S \setminus M.P'$ behaves similarly to P but is absorbed by the first marked state it hits. Consider the quantization $W_{P'}$ of P' and define the *quantum hitting time*, H, of set M to be the smallest m for which $|W_{P'}^m \phi_0 - \phi_0| \geq 0.1$, where $\phi_0 := \sum_{x \in S} \sqrt{1/N}|x\rangle|p_x\rangle$ (which is stationary for W_P). Note that the construction cost of $W_{P'}$ is proportional to $U + C$.

Why is this definition of quantum hitting time interesting? The classical hitting time measures the number of iterations of the absorbing walk P' required to noticeably skew the uniform starting distribution. Similarly, the quantum hitting time bounds the number of iterations of the following quantum algorithm for detecting whether M is nonempty: At each step, apply operator $W_{P'}$. If M is empty, then $P' = P$ and the starting state is left invariant. If M is nonempty, then the angle between $W^t_{P'}\phi_0$ and $W^t_P\phi_0$ gradually increases. Using an additional *control register* to apply either $W_{P'}$ or W_P with quantum control, the divergence of these two states (should M be nonempty) can be detected. The required number of iterations is exactly H.

It remains to compute H. When P is symmetric and *ergodic*, the expression for the classical hitting time has a quantum analogue [18] ($|M| \leq N/2$ for technical reasons):

$$H \leq \sum_{k=1}^{N-|M|} \frac{\nu_k^2}{\sqrt{1-\lambda_k}}\,, \qquad (1)$$

where ν_k is the sum of the coordinates of v_k divided by $1/\sqrt{N}$. From (1) and the expression for h one can derive an amazing connection between the classical and quantum hitting times:

Theorem 2 ([18]) *Let P be symmetric and ergodic, and let h be the classical hitting time for marked set M and uniform starting distribution. Then the quantum hitting time of M is at most $\sqrt{h}$.*

One can further show:

Theorem 3 ([18]) *If P is state-transitive and $|M| = 1$, then the marked state is observed with probability at least N/h in $O(\sqrt{h})$ steps.*

Theorems 2 and 3 imply most quantum hitting time results of the previous section, *without calculation*, relying only on estimates of the corresponding classical hitting times. Expression (1) is based on a fundamental connection between the eigenvalues and eigenvectors of P and W_P. Notice that R_1 and R_2 are reflections on the subspaces generated by $\{|p_x\rangle \otimes |x\rangle |\, x \in S\}$ and $\{|x\rangle \otimes |p_x\rangle |\, x \in S\}$, respectively. Hence the eigenvalues of R_1R_2 can be expressed in terms of the eigenvalues of the mutual Gram matrix of these systems. This matrix D, the *discriminant matrix* of P, is:

$$D(P) = \sqrt{P \circ P^T} \overset{\text{def}}{=} (\sqrt{p_{x,y}p_{y,x}})_{x,y\in S}\,. \qquad (2)$$

If P is symmetric then $D(P) = P$, and the formula remains fairly simple even when P is not symmetric. In particular, the absorbing walk P' has discriminant matrix $\left[\begin{smallmatrix} P_M & 0 \\ 0 & I \end{smallmatrix}\right]$. Finally, the relation between $D(P)$ and the spectral decomposition of W_P is given by:

Theorem 4 ([18]) *Let P be an arbitrary Markov chain on a finite state space S and let $\cos\theta_1 \geq \cdots \geq \cos\theta_l$ be those singular values of $D(P)$ lying in the open interval $(0,1)$, with associated singular vector pairs v_j, w_j for $1 \leq j \leq l$. Then the non-trivial eigenvalues of W_P (excluding 1 and −1) and their corresponding eigenvectors are $e^{-2i\theta_j}$, $R_1w_j - e^{-i\theta_j}R_2v_j$; $e^{2i\theta_j}$ and $R_jw_j - e^{i\theta_j}R_2v_j$ for $1 \leq j \leq l$.*

Latest Development

Recently, Magniez et al. [12] have used Szegedy's quantization W_P of an ergodic walk P (rather than its absorbing version P') to obtain an efficient and general implementation of the abstract search algorithm of Ambainis et al. [4].

Theorem 5 ([12]) *Let P be reversible and ergodic with spectral gap $\delta > 0$. Let M have marked probability either zero or $\varepsilon > 0$. Then there is a quantum algorithm solving the hitting problem (search version) with cost $S + \frac{1}{\sqrt{\varepsilon}}\left(\frac{1}{\sqrt{\delta}}U + C\right)$.*

Applications

Element Distinctness

Suppose one is given elements $x_1, \ldots, x_m \in \{1, \ldots, m\}$ and is asked if there exist i, j such that $x_i = x_j$. The classical query complexity of this problem is $\Theta(m)$. Ambainis [2] gave an (optimal) $O(m^{2/3})$ quantum query algorithm using a quantum walk on the Johnson graph of $m^{2/3}$-subsets of $\{1, \ldots, m\}$ with those subsets containing i, j with $x_i = x_j$ marked.

Triangle Finding

Suppose one is given the adjacency matrix A of a graph on n vertices and is required to determine if the graph contains a triangle (i. e., a clique of size 3) using as few queries as possible to the entries of A. The classical query complexity of this problem is $\Theta(n^2)$. Magniez, Santha, and Szegedy [13] gave an $\tilde{O}(n^{1.3})$ algorithm by adapting [2]. This was improved to $O(n^{1.3})$ by Magniez et al. [12].

Matrix Product Verification

Suppose one is given three $n \times n$ matrices A, B, C and is required to determine if $AB \neq C$ (i. e., if their exist i, j such that $\sum_k A_{ik}B_{kj} \neq C_{ij}$) using as few queries as possible to the entries of A, B, and C. This problem has classical query complexity $\Theta(n^2)$. Buhrman and Spalek [5] gave an $O(n^{5/3})$ quantum query algorithm using [18].

Group Commutativity Testing

Suppose one is presented with a black-box group specified by its k generators and is required to determine if the group commutes using as few queries as possible to the group product operation (i. e., queries of the form "What is the product of elements g and h?"). The classical query complexity is $\Theta(k)$ group operations. Magniez and Nayak [11] gave an (essentially optimal) $\tilde{O}(k^{2/3})$ quantum query algorithm by walking on the product of two graphs whose vertices are (ordered) l-tuples of distinct generators and whose transition probabilities are nonzero only where the l-tuples at two endpoints differ in at most one coordinate.

Open Problems

Many issues regarding quantization of Markov chains remain unresolved, both for the hitting problem and the closely related mixing problem.

Hitting

Can the quadratic quantum hitting time speedup be extended from all symmetric Markov chains to all reversible ones? Can the lower bound of [18] on observation probability be extended beyond the class of state-transitive Markov chains with a unique marked state? What other algorithmic applications of quantum hitting time can be found?

Mixing

Another wide use of Markov chains in classical algorithms is in *mixing*. In particular, *Markov chain Monte Carlo* algorithms work by running an ergodic Markov chain with carefully chosen stationary distribution π until reaching its *mixing time*, at which point the current state is guaranteed to be distributed ε-close to uniform. Such algorithms form the basis of most randomized algorithms for approximating #P-complete problems. Hence, the problem is:

INPUT: Markov chain P on set S, tolerance $\varepsilon > 0$.
OUTPUT: A state ε-close to π in total variation distance.

Notions of quantum mixing time were first proposed and analyzed on the line, the cycle, and the hypercube by Nayak et al. [3,15], Aharonov et al. [1], and Moore and Russell [14]. Recent work of Kendon and Tregenna [10] and Richter [16] has investigated the use of decoherence in improving mixing of quantum walks. Two fundamental questions about the quantum mixing time remain open: What is the "most natural" definition? And, when is there a quantum speedup over the classical mixing time?

Cross References

► Quantum Algorithm for Element Distinctness
► Quantum Algorithm for Finding Triangles

Recommended Reading

1. Aharonov, D., Ambainis, A., Kempe, J., Vazirani, U.: Quantum walks on graphs. In: Proc. STOC (2001)
2. Ambainis, A.: Quantum walk algorithm for element distinctness. SIAM J. Comput. **37**(1), 210–239 (2007). Preliminary version in Proc. FOCS 2004
3. Ambainis, A., Bach, E., Nayak, A., Vishwanath, A., Watrous, J.: One-dimensional quantum walks. In: Proc. STOC (2001)
4. Ambainis, A., Kempe, J., Rivosh, A.: Coins make quantum walks faster. In: Proc. SODA (2005)
5. Buhrman, H., Spalek, R.: Quantum verification of matrix products. In: Proc. SODA (2006)
6. Childs, A., Cleve, R., Deotto, E., Farhi, E., Gutmann, S., Spielman, D.: Exponential algorithmic speedup by a quantum walk. In: Proc. STOC (2003)
7. Farhi, E., Gutmann, S.: Quantum computation and decision trees. Phys. Rev. A **58** (1998)
8. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: Proc. STOC (1996)
9. Kempe, J.: Discrete quantum walks hit exponentially faster. In: Proc. RANDOM (2003)
10. Kendon, V., Tregenna, B.: Decoherence can be useful in quantum walks. Phys. Rev. A. **67**, 42–315 (2003)
11. Magniez, F., Nayak, A.: Quantum complexity of testing group commutativity. Algorithmica **48**(3), 221–232 (2007) Preliminary version in Proc. ICALP 2005
12. Magniez, F., Nayak, A., Roland, J., Santha, M.: Search via quantum walk. In: Proc. STOC (2007)
13. Magniez, F., Santha, M., Szegedy, M.: Quantum algorithms for the triangle problem. SIAM J. Comput. **37**(2), 413–424 (2007) Preliminary version in Proc. SODA 2005
14. Moore, C., Russell, A.: Quantum walks on the hypercube. In: Proc. RANDOM (2002)
15. Nayak, A., Vishwanath, A.: Quantum walk on the line. quant-ph/0010117
16. Richter, P.C.: Quantum speedup of classical mixing processes. Phys. Rev. A **76**, 042306 (2007)
17. Shenvi, N., Kempe, J., Whaley, K.B.: A quantum random walk search algorithm. Phys. Rev. A **67**, 52–307 (2003)
18. Szegedy, M.: Quantum speed-up of Markov chain based algorithms. In: Proc. FOCS (2004)

Quantum Algorithm for Checking Matrix Identities

2006; Buhrman, Spalek

ASHWIN NAYAK
Department of Combinatorics and Optimization, University of Waterloo and Perimeter Institute for Theoretical Physics, Waterloo, ON, Canada

Keywords and Synonyms

Matrix product verification

Problem Definition

Let A, B, C be three given matrices of dimension $n \times n$ over a field, where C is claimed to be the matrix product AB. The straightforward method of checking whether $C = AB$ is to multiply the matrices A, B, and compare the entries of the result with those of C. This takes time $O(n^{\omega})$, where ω is the "exponent of matrix multiplication". It is evident from the definition of the matrix multiplication operation that $2 \le \omega \le 3$. The best known bound on ω is 2.376 [4].

Here, and in the sequel, "time" is taken to mean "number of arithmetic operations" over the field (or other algebraic structure to which the entries of the matrix belong). Similarly, in stating space complexity, the multiplicative factor corresponding to the space required to represent elements of the algebraic structure is suppressed.

Surprisingly, matrix multiplication can be circumvented by using a randomized "fingerprinting" technique due to Freivalds [5], and the matrix product can be checked in time $O(n^2)$ with one-sided bounded probability of error. This algorithm extends, in fact, to matrices over any *integral domain* [3] and the number of random bits used may be reduced to $\log \frac{n}{\epsilon} + O(1)$ for an algorithm that makes one-sided probabilistic error at most ϵ [8]. (All logarithms in this article are taken to base 2.) The fingerprinting technique has found numerous other applications in theoretical computer science (see, for example [10]).

Buhrman and Špalek consider the complexity of checking matrix products on a quantum computer.

Problem 1 (Matrix product verification)
INPUT: *Matrices A, B, C of dimension $n \times n$ over an integral domain.*
OUTPUT: EQUAL *if $C = AB$, and* NOT EQUAL *otherwise.*

They also study the verification problem over the Boolean algebra $\{0, 1\}$ with operations $\{\vee, \wedge\}$, where the fingerprinting technique does not apply.

As an application of their verification algorithms, they consider multiplication of sparse matrices.

Problem 2 (Matrix multiplication)
INPUT: *Matrices A, B of dimension $n \times n$ over an integral domain or the Boolean algebra $\{0, 1\}$.*
OUTPUT: *The matrix product $C = AB$ over the integral domain or the Boolean algebra.*

Key Results

Ambainis, Buhrman, Høyer, Karpinski, and Kurur [2] first studied matrix product verification in the quantum mechanical setting. Using a recursive application of the Grover search algorithm [6], they gave an $O(n^{7/4})$ algorithm for the problem. Buhrman and Špalek improve this runtime by adapting search algorithms based on quantum walk that were recently discovered by Ambainis [1] and Szegedy [11].

Let $W = \{(i, j) | (AB - C)_{i,j} \neq 0\}$ be the set of coordinates where C disagrees with the product AB, and let W' be the largest independent subset of W. (A set of coordinates is said to be *independent* if no row or column occurs more than once in the set.) Define $q(W) = \max\{|W'|, \min\{|W|, \sqrt{n}\}\}$.

Theorem 1 *Consider Problem 1. There is a quantum algorithm that always returns* EQUAL *if $C = AB$, returns* NOT EQUAL *with probability at least 2/3 if $C \neq AB$, and has worst case run-time $O(n^{5/3})$, expected run-time $O(n^{2/3}/q(W)^{1/3})$, and space complexity $O(n^{5/3})$.*

Buhrman and Špalek state their results in terms of "black-box" complexity or "query complexity", where the entries of the input matrices A, B, C are provided by an oracle. The measure of complexity here is the number of oracle calls (queries) made. The query complexity of their quantum algorithm is the same as the run time in the above theorem. They also derive a lower bound on the query complexity of the problem.

Theorem 2 *Any bounded-error quantum algorithm for Problem 1 has query complexity $\Omega(n^{3/2})$.*

When the matrices A, B, C are Boolean, and the product is defined over the operations $\{\vee, \wedge\}$, an optimal algorithm with run-time/query complexity $O(n^{3/2})$ may be derived from an algorithm for AND-OR trees [7]. This has space complexity $O((\log n)^3)$.

All the quantum algorithms may be generalized to handle rectangular matrix product verification, with appropriate modification to the run-time and space complexity.

Applications

Using binary search along with the algorithms in the previous section, the position of a wrong entry in a matrix C (purported to be the product AB) can be located, and then corrected. Buhrman and Špalek use this in an iterative fashion to obtain a matrix multiplication algorithm, starting from the guess $C = 0$. When the product AB is a sparse matrix, this leads to a quantum matrix multiplication scheme that is, for some parameters, faster than known classical schemes.

Theorem 3 *For any $n \times n$ matrices $A\,B$ over an integral domain, the matrix product $C = AB$ can be computed by a quantum algorithm with polynomially small error probability in expected time*

$$O(1) \cdot \begin{cases} n \log n \cdot n^{2/3} w^{2/3} & \text{when } 1 \le w \le \sqrt{n}\,, \\ n \log n \cdot \sqrt{n}\, w & \text{when } \sqrt{n} \le w \le n\,, \text{ and} \\ n \log n \cdot n\sqrt{w} & \text{when } n \le w \le n^2\,, \end{cases}$$

where w is the number of non-zero entries in C.

A detailed comparison of this quantum algorithm with classical ones may be found in [3].

A subsequent quantum walk based algorithm due to Magniez, Nayak, Roland, and Santha [9] finds a wrong entry in the same run-time as in Theorem 1, without the need for binary search. This improves the run-time of the quantum algorithm for matrix multiplication described above slightly.

Since Boolean matrix products can be verified faster, boolean matrix products can be computed in expected time $O(n^{3/2} w)$, where w is the number of '1' entries in the product.

All matrix product algorithms presented here may be used for multiplication of rectangular matrices as well, with appropriate modifications.

Cross References

► Quantization of Markov Chains
► Quantum Algorithm for Element Distinctness

Recommended Reading

1. Ambainis, A.: Quantum walk algorithm for Element Distinctness. In: Proceedings of the 45th Symposium on Foundations of Computer Science, pp. 22–31, Rome, Italy, 17–19 October 2004
2. Ambainis, A., Buhrman, H., Høyer, P., Karpinski, M., Kurur, P.: Quantum matrix verification. Unpublished manuscript (2002)
3. Buhrman, H., Špalek, R.: Quantum verification of matrix products. In: Proceedings of 17th ACM-SIAM Symposium on Discrete Algorithms, pp. 880–889, Miami, FL, USA, 22–26 January 2006
4. Coppersmith, D., Winograd, S.: Matrix multiplication via arithmetic progressions. J. Symb. Comput. **9**(3), 251–280 (1990)
5. Freivalds, R.: Fast probabilistic algorithms. In: Proceedings of the 8th Symposium on Mathematical Foundations of Computer Science, pp. 57–69, Olomouc, Czechoslovakia, 3–7 September 1979
6. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: Proceedings of the 28th ACM Symposium on the Theory of Computing, pp. 212–219, Philadelphia, PA, USA, 22–24 May 1996
7. Høyer, P., Mosca, M., de Wolf, R.: Quantum search on bounded-error inputs. In: Proceedings of the 30th International Colloquium on Automata, Languages and Programming. Lecture Notes in Computer Science, vol. 2719, pp. 291–299, Eindhoven, The Netherlands, 30 June – 4 July 2003
8. Kimbrel, T., Sinha, R.K.: A probabilistic algorithm for verifying matrix products using $O(n^2)$ time and $\log_2 n + O(1)$ random bits. Inf. Proc. Lett. **45**(2), 107–110 (1993)
9. Magniez, F., Nayak, A., Roland, J., Santha, M.: Search via quantum walk. In: Proceeding of the 39th ACM Symposium on Theory of Computing, pp. 575–584, San Diego, CA, USA, 11–13 June 2007 (2007)
10. Motwani, R., Raghavan, P.: Randomized Algorithms. Cambridge University Press, Cambridge (1995)
11. Szegedy, M.: Quantum speed-up of Markov chain based algorithms. In: Proceedings of the 45th IEEE Symposium on Foundations of Computer Science, pp. 32–41, Rome, Italy, 17–19 October 2004 (2004)

Quantum Algorithm for the Collision Problem

1998; Brassard, Hoyer, Tapp

ALAIN TAPP
DIRO, University of Montréal, Montreal, QC, Canada

Problem Definition

A function F is said to be *r-to-one* if every element in its image has exactly r distinct preimages.
Input: an r-to-one function F.
Output: x_1 and x_2 such that $F(x_1) = F(x_2)$.

Key Results

The algorithm presented here finds collisions in arbitrary r-to-one functions F after only $O(\sqrt[3]{N/r})$ expected evaluations of F. The algorithm uses the function as a black box, that is, the only thing the algorithm requires is the capacity to evaluate the function. Again assuming the function is given by a black box, the algorithm is optimal [1] and it is more efficient than the best *possible* classical algorithm, which has query complexity $\Omega(\sqrt{N/r})$. The result is stated precisely in the following theorem and corollary.

Theorem 1 *Given an r-to-one function $F\colon X \to Y$ with $r \ge 2$ and an integer $1 \le k \le N = |X|$, algorithm* **Collision**(F, k) *returns a collision after an expected number of $O(k + \sqrt{N/(rk)})$ evaluations of F and uses space $\Theta(k)$. In particular, when $k = \sqrt[3]{N/r}$ then* **Collision**(F, k) *uses an expected number of $O(\sqrt[3]{N/r})$ evaluations of F and space $\Theta(\sqrt[3]{N/r})$.*

Corollary 2 *There exists a quantum algorithm that can find a collision in an arbitrary r-to-one function* $F\colon X \to Y$*, for any* $r \geq 2$*, using space* S *and an expected number of* $O(T)$ *evaluations of* F *for every* $1 \leq S \leq T$ *subject to*

$$ST^2 \geq |F(X)|,$$

where $F(X)$ *denotes the image of* F.

The algorithm uses as a procedure a version of Grover's search algorithm. Given a function H with domain size n and a target y, **Grover**(H, y) returns an x such that $H(x) = y$ in expected $O(\sqrt{n})$ evaluations of H.

Collision(F,k)

1. Pick an arbitrary subset $K \subseteq X$ of cardinality k. Construct a table L of size k where each item in L holds a distinct pair $(x, F(x))$ with $x \in K$.
2. Sort L according to the second entry in each item of L.
3. Check if L contains a collision, that is, check if there exist distinct elements $(x_0, F(x_0)), (x_1, F(x_1)) \in L$ for which $F(x_0) = F(x_1)$. If so, go to step 6.
4. Compute $x_1 =$ **Grover**$(H, 1)$, where $H : X \to \{0, 1\}$ denotes the function defined by $H(x) = 1$ if and only if there exists $x_0 \in K$ so that $(x_0, F(x)) \in L$ but $x \neq x_0$. (Note that x_0 is unique if it exists since we already checked that there are no collisions in L.)
5. Find $(x_0, F(x_1)) \in L$.
6. Output the collision $\{x_0, x_1\}$.

Applications

This problem is of particular interest for cryptology because some functions known as *hash functions* are used in various cryptographic protocols. The security of these protocols crucially depends on the presumed difficulty of finding collisions in such functions.

Cross References

► Greedy Set-Cover Algorithms

Recommended Reading

1. Aaronson, S., Shi, Y.: Quantum Lower Bounds for the Collision and the Element Distinctness Problems. J. ACM **51**(4), 595–605 (2004)
2. Boyer, M., Brassard, G., Høyer, P., Tapp A.: Tight bounds on quantum searching. Fortschr. Phys. **46**(4–5), 493–505 (1998)
3. Brassard, G., Høyer, P., Mosca, M., Tapp A.: Quantum Amplitude Amplification and Estimation. In: Lomonaco, S.J. (ed.) Quantum Computation & Quantum Information Science, AMS Contemporary Mathematics Series Millennium Volume, vol. 305, pp. 53–74. American Mathematical Society, Providence (2002)
4. Brassard, G., Høyer, P., Tapp, A.: Quantum Algorithm for the Collision Problem. 3rd Latin American Theoretical Informatics Symposium (LATIN'98). LNCS, vol. 1380, pp. 163–169. Springer (1998)
5. Carter, J.L., Wegman, M.N.: Universal classes of hash functions. J. Comput. Syst. Sci. **18**(2), 143–154 (1979)
6. Grover, L.K.: A fast quantum mechanical algorithm for database search. Proceedings of the 28th Annual ACM Symposium on Theory of Computing, pp. 212–219. ACM (1996)
7. Stinson, D.R.: Cryptography: Theory and Practice, CRC Press, Inc (1995)
8. Nielsen, M.A., Chuang, I.L.: Quantum Computation and Quantum Information. Cambridge University Press, Cambridge (2000)

Quantum Algorithm for the Discrete Logarithm Problem

1994; Shor

Pranab Sen
School of Technology and Computer Science,
Tata Institute of Fundamental Research,
Mumbai, India

Keywords and Synonyms

Logarithms in groups

Problem Definition

Given positive real numbers $a \neq 1, b$, the logarithm of b to base a is the unique real number s such that $b = a^s$. The notion of the *discrete logarithm* is an extension of this concept to general groups.

Problem 1 (Discrete logarithm)
Input: *Group* G*,* $a, b \in G$ *such that* $b = a^s$ *for some positive integer* s*.*
Output: *The smallest positive integer* s *satisfying* $b = a^s$*, also known as the discrete logarithm of* b *to the base* a *in* G*.*

The usual logarithm corresponds to the discrete logarithm problem over the group of positive reals under multiplication. The most common case of the discrete logarithm problem is when the group $G = \mathbb{Z}_p^*$, the multiplicative group of integers between 1 and $p - 1$ modulo p, where p is a prime. Another important case is when the group G is the group of points of an elliptic curve over a finite field.

Key Results

The discrete logarithm problem in $\mathbb{Z}_p^*$, where p is a prime, as well as in the group of points of an elliptic curve over a finite field is believed to be intractable for randomized classical computers. That is any, possibly randomized, algorithm for the problem running on a classical computer will take time that is superpolynomial in the number of bits required to describe an input to the problem. The best classical algorithm for finding discrete logarithms in $\mathbb{Z}_p^*$, where p is a prime, is Gordon's [4] adaptation of the number field sieve which runs in time $\exp(O((\log p)^{1/3}(\log\log p)^{2/3}))$.

In a breakthrough result, Shor [9] gave an efficient quantum algorithm for the discrete logarithm problem in any group G; his algorithm runs in time that is polynomial in the bit size of the input.

Result 1 **([9])** *There is a quantum algorithm solving the discrete logarithm problem in any group G on n-bit inputs in time $O(n^3)$ with probability at least 3/4.*

Description of the Discrete Logarithm Algorithm

Shor's algorithm [9] for the discrete logarithm problem makes essential use of an efficient quantum procedure for implementing a unitary transformation known as the *quantum Fourier transform*. His original algorithm gave an efficient procedure for performing the quantum Fourier transform only over groups of the form $\mathbb{Z}_r$, where r is a "smooth" integer, but nevertheless, he showed that this itself sufficed to solve the discrete logarithm in the general case. In this article, however, a more modern description of Shor's algorithm is given. In particular, a result by Hales and Hallgren [5] is used which shows that the quantum Fourier transform over any finite cyclic group $\mathbb{Z}_r$ can be efficiently approximated to inverse-exponential precision.

A description of the algorithm is given below. A general familiarity with quantum notation on the part of the reader is assumed. A good introduction to quantum computing can be found in the book by Nielsen and Chuang [8]. Let $(G, a, b, \bar{r})$ be an instance of the discrete logarithm problem, where $\bar{r}$ is a supplied upper bound on the order of a in G. That is, there exists a positive integer $r \le \bar{r}$ such that $a^r = 1$. By using an efficient quantum algorithm for order finding also discovered by Shor [9], one can assume that the order of a in G is known, that is, the smallest positive integer r satisfying $a^r = 1$. Shor's order-finding algorithm runs in time $O((\log \bar{r})^3)$. Let $\epsilon > 0$. The discrete logarithm algorithm works on three registers, of which the first two are each t qubits long, where $t := O(\log r + \log(1/\epsilon))$, and the third register is big enough to store an element of G. Let U denote the unitary transformation

$$U\colon |x\rangle|y\rangle|z\rangle \mapsto |x\rangle|y\rangle|z \oplus (b^x a^y)\rangle \;,$$

where $\oplus$ denotes bitwise XOR. Given access to a reversible oracle for group operations in G, U can be implemented reversibly in time $O(t^3)$ by repeated squaring.

Let $\mathbb{C}[\mathbb{Z}_r]$ denote the Hilbert space of functions from $\mathbb{Z}_r$ to complex numbers. The computational basis of $\mathbb{C}[\mathbb{Z}_r]$ consists of the delta functions $\{|l\rangle\}_{0\le l\le r-1}$, where $|\rangle$ is the function that sends the element l to 1 and the other elements of $\mathbb{Z}_r$ to 0. Let $\mathrm{QFT}_{\mathbb{Z}_r}$ denote the *quantum Fourier transform* over the cyclic group $\mathbb{Z}_r$ defined as the following unitary operator on $\mathbb{C}[\mathbb{Z}_r]$:

$$\mathrm{QFT}_{\mathbb{Z}_r}\colon |x\rangle \mapsto r^{-1/2} \sum_{y\in\mathbb{Z}_r} \mathrm{e}^{-2\pi i x y/r}|y\rangle \;.$$

It can be implemented in quantum time $O(t\log(t/\epsilon) + \log^2(1/\epsilon))$ up to an error of ϵ using one t-qubit register [5]. Note that for any $k \in \mathbb{Z}_r$, $\mathrm{QFT}_{\mathbb{Z}_r}$ transforms the state $r^{-1/2}\sum_{x\in\mathbb{Z}_r} \mathrm{e}^{2\pi i k x/r}|x\rangle$ to the state $|k\rangle$. For any integer l, $0 \le l \le r-1$, define

$$|\hat{l}\rangle := r^{-1/2} \sum_{k=0}^{r-1} \mathrm{e}^{-2\pi i l k/r}|a^k\rangle \;. \tag{1}$$

Observe that $\{|\hat{l}\rangle\}_{0\le l\le r-1}$ forms an orthonormal basis of $\mathbb{C}[\langle a\rangle]$, where $\langle a\rangle$ is the subgroup generated by a in G and is isomorphic to $\mathbb{Z}_r$, and $\mathbb{C}[\langle a\rangle]$ denotes the Hilbert space of functions from $\langle a\rangle$ to complex numbers.

Algorithm 1 (Discrete logarithm)
Input: Elements $a, b \in G$, a quantum circuit for U, the order r of a in G.
Output: With constant probability, the discrete logarithm s of b to the base a in G.
Runtime: A total of $O(t^3)$ basic gate operations, including four invocations of $\mathrm{QFT}_{\mathbb{Z}_r}$ and one of U.

PROCEDURE:
1. Repeat Steps (a)–(e) twice, obtaining $(sl_1 \bmod r, l_1)$ and $(sl_2 \bmod r, l_2)$.

```
(a)  $|0\rangle|0\rangle|0\rangle$
     Initialization;
(b)  $\mapsto r^{-1} \sum_{x,y\in\mathbb{Z}_r} |x\rangle|y\rangle|0\rangle$
     ApplyQFT$_{\mathbb{Z}_r}$to the first two registers;
(c)  $\mapsto r^{-1} \sum_{x,y\in\mathbb{Z}_r} |x\rangle|y\rangle|b^x a^y\rangle$
     Apply $U$
```

$$(d) \quad \mapsto r^{-1/2} \sum_{l=0}^{r-1} |sl \bmod r\rangle |l\rangle |\hat{l}\rangle$$

Apply $\mathrm{QFT}_{\mathbb{Z}_r}$ to the first two registers;

$$(e) \quad \mapsto (sl \bmod r, l)$$

Measure the first two registers;

2. If l_1 is not coprime to l_2, abort.
3. Let k_1, k_2 be integers such that $k_1 l_1 + k_2 l_2 = 1$. Then, output $s = k_1(sl_1) + k_2(sl_2) \bmod r$.

The working of the algorithm is explained below. From Eq. (1), it is easy to see that

$$|b^x a^y\rangle = r^{-1/2} \sum_{l=0}^{r-1} e^{2\pi i l(sx+y)/r} |\hat{l}\rangle \,.$$

Thus, the state in Step 1(c) of the above algorithm can be written as

$$\begin{aligned} &r^{-1} \sum_{x,y \in \mathbb{Z}_r} |x\rangle |y\rangle |b^x a^y\rangle \\ &= r^{-3/2} \sum_{l=0}^{r-1} \sum_{x,y\in\mathbb{Z}_r} e^{2\pi i l(sx+y)/r} |x\rangle |y\rangle |\hat{l}\rangle \\ &= r^{-3/2} \sum_{l=0}^{r-1} \left[\sum_{x\in\mathbb{Z}_r} e^{2\pi i s l x/r} |x\rangle \right] \cdot \left[\sum_{y\in\mathbb{Z}_r} e^{2\pi i l y/r} |y\rangle \right] |\hat{l}\rangle . \end{aligned}$$

Now, applying $\mathrm{QFT}_{\mathbb{Z}_r}$ to the first two registers gives the state in Step 1(d) of the above algorithm. Measuring the first two registers gives $(sl \bmod r, l)$ for a uniformly distributed $l, 0 \le l \le r-1$ in Step 1(e). By elementary number theory, it can be shown that if integers l_1, l_2 are uniformly and independently chosen between 0 and $l-1$, they will be coprime with constant probability. In that case, there will be integers k_1, k_2 such that $k_1 l_1 + k_2 l_2 = 1$, leading to the discovery of the discrete logarithm s in Step 3 of the algorithm with constant probability. Since actually only an ϵ-approximate version of $\mathrm{QFT}_{\mathbb{Z}_r}$ can be applied, ϵ can be set to be a sufficiently small constant, and this will still give the correct discrete logarithm s in Step 3 of the algorithm with constant probability. The success probability of Shor's algorithm for the discrete logarithm problem can be boosted to at least 3/4 by repeating it a constant number of times.

Generalizations of the Discrete Logarithm Algorithm

The discrete logarithm problem is a special case of a more general problem called the *hidden subgroup problem* [8]. The ideas behind Shor's algorithm for the discrete logarithm problem can be generalized in order to yield an efficient quantum algorithm for hidden subgroups in Abelian groups (see [1] for a brief sketch). It turns out that finding the discrete logarithm of b to the base a in G reduces to the hidden subgroup problem in the group $\mathbb{Z}_r \times \mathbb{Z}_r$ where r is the order of a in G. Besides the discrete logarithm problem, other cryptographically important functions like integer factoring, finding the order of permutations, as well as finding self-shift-equivalent polynomials over finite fields can be reduced to instances of a hidden subgroup in Abelian groups.

Applications

The assumed intractability of the discrete logarithm problem lies at the heart of several cryptographic algorithms and protocols. The first example of public-key cryptography, namely, the Diffie–Hellman key exchange [2], uses discrete logarithms, usually in the group $\mathbb{Z}_p^*$ for a prime p. The security of the US national standard Digital Signature Algorithm (see [7] for details and more references) depends on the assumed intractability of discrete logarithms in $\mathbb{Z}_p^*$, where p is a prime. The ElGamal public-key cryptosystem [3] and its derivatives use discrete logarithms in appropriately chosen subgroups of $\mathbb{Z}_p^*$, where p is a prime. More recent applications include those in elliptic curve cryptography [6], where the group consists of the group of points of an elliptic curve over a finite field.

Cross References

► Abelian Hidden Subgroup Problem
► Quantum Algorithm for Factoring

Recommended Reading

1. Brassard, G., Høyer, P.: An exact quantum polynomial-time algorithm for Simon's problem. In: Proceedings of the 5th Israeli Symposium on Theory of Computing and Systems, pp. 12–23, Ramat-Gan, 17–19 June 1997
2. Diffie, W., Hellman, M.: New directions in cryptography. IEEE Trans. Inf. Theor. **22**, 644–654 (1976)
3. ElGamal, T.: A public-key cryptosystem and a signature scheme based on discrete logarithms. IEEE Trans. Inf. Theor. **31**(4), 469–472 (1985)
4. Gordon, D.: Discrete logarithms in GF(p) using the number field sieve. SIAM J. Discret. Math. **6**(1), 124–139 (1993)
5. Hales, L., Hallgren, S.: An improved quantum Fourier transform algorithm and applications. In: Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science, pp. 515–525 (2000)
6. Hankerson, D., Menezes, A., Vanstone, S.: Guide to Elliptic Curve Cryptography. Springer, New York (2004)

7. Menezes, A., van Oorschot, P., Vanstone, S.: Handbook of Applied Cryptography. CRC Press, Boca Raton (1997)
8. Nielsen, M., Chuang, I.: Quantum computation and quantum information. Cambridge University Press, Cambridge (2000)
9. Shor, P.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM J. Comput. **26**(5), 1484–1509 (1997)

Quantum Algorithm for Element Distinctness

2004; Ambainis

ANDRIS AMBAINIS
Department of Computer Science, University of Latvia, Riga, Latvia

Problem Definition

In the *element distinctness* problem, one is given a list of N elements $x_1, \dots, x_N \in \{1, \dots, m\}$ and one must determine if the list contains two equal elements. Access to the list is granted by submitting queries to a black box, and there are two possible types of query.

Value queries. In this type of query, the input to the black box is an index i. The black box outputs x_i as the answer. In the quantum version of this model, the input is a quantum state that may be entangled with the workspace of the algorithm. The joint state of the query, the answer register, and the workspace may be represented as $\sum_{i,y,z} a_{i,y,z}|i, y, z\rangle$, with y being an extra register which will contain the answer to the query and z being the workspace of the algorithm. The black box transforms this state into $\sum_{i,y,z} a_{i,y,z}|i, (y + x_i) \bmod m, z\rangle$. The simplest particular case is if the input to the black box is of the form $\sum_i a_i|i, 0\rangle$. Then, the black box outputs $\sum_i a_i|i, x_i\rangle$. That is, a quantum state consisting of the index i is transformed into a quantum state, each component of which contains x_i together with the corresponding index i.

Comparison queries. In this type of query, the input to the black box consists of two indices i, j. The black box gives one of three possible answers: "$x_i > x_j$", "$x_i < x_j$" or "$x_i = x_j$". In the quantum version, the input is a quantum state consisting of basis states $|i, j, z\rangle$, with i, j being two indices and z being algorithm's workspace.

There are several reasons why the element distinctness problem is interesting to study. First of all, it is related to sorting. Being able to sort $x_1, \dots, x_N$ enables one to solve the element distinctness by first sorting $x_1, \dots, x_N$ in increasing order. If there are two equal elements $x_i = x_j$, then they will be next one to another in the sorted list. Therefore, after one has sorted $x_1, \dots, x_N$, one must only check the sorted list to see if each element is different from the next one. Because of this relation, the element distinctness problem might capture some of the same difficulty as sorting. This has lead to a long line of research on classical lower bounds for the element distinctness problem (cf [6,8,15]. and many other papers).

Second, the central concept of the algorithms for the element distinctness problem is the notion of a collision. This notion can be generalized in different ways, and its generalizations are useful for building quantum algorithms for various graph-theoretic problems (e. g. triangle finding [12]) and matrix problems (e. g. checking matrix identities [7]).

A generalization of element distinctness is element k-distinctness [2], in which one must determine if there exist k different indices $i_1, \dots, i_k \in \{1, \dots, N\}$ such that $x_{i_1} = x_{i_2} = \dots = x_{i_k}$. A further generalization is the k-subset finding problem [9], in which one is given a function $f(y_1, \dots, y_k)$, and must determine whether there exist $i_1, \dots, i_k \in \{1, \dots, N\}$ such that $f(x_{i_1}, x_{i_2}, \dots, x_{i_k}) = 1$.

Key Results

Element Distinctness: Summary of Results

In the classical (non-quantum) context, the natural solution to the element distinctness problem is done by sorting, as described in the previous section. This uses $O(N)$ value queries (or $O(N \log N)$ comparison queries) and $O(N \log N)$ time. Any classical algorithm requires $\Omega(N)$ value or $\Omega(N \log N)$ comparison queries. If the algorithm is restricted to $o(N)$ space, stronger lower bounds are known [15].

In the quantum context, Buhrman et al. [5] gave the first non-trivial quantum algorithm, using $O(N^{3/4})$ queries. Ambainis [2] then designed a new algorithm, based on a novel idea using quantum walks. Ambainis' algorithm uses $O(N^{2/3})$ queries and is known to be optimal: Aaronson and Shi [1,3,10] have shown that any quantum algorithm for element distinctness must use $\Omega(N^{2/3})$ queries.

For quantum algorithms that are restricted to storing r values x_i (where $r < N^{2/3}$), the best algorithm runs in $O(N/\sqrt{r})$ time.

All of these results are for value queries. They can be adapted to the comparison query model, with an $\log N$ factor increase in the complexity. The time complexity is within a polylogarithmic $O(\log^c N)$ factor of the query complexity, as long as the computational model is sufficiently general [2]. (Random access quantum memory is necessary for implementing any of the two known quantum algorithms.)

Element k-distinctness (and k-subset finding) can be solved with $O(N^{k/(k+1)})$ value queries, using $O(N^{k/(k+1)})$ memory. For the case when the memory is restricted to $r < N^{k/(k+1)}$ values of x_i, it suffices to use $O(r + (N^{k/2})/(r^{(k-1)/2}))$ value queries. The results generalize to comparison queries and time complexity, with a polylogarithmic factor increase in the time complexity (similarly to the element distinctness problem).

Element Distinctness: The Methods

Ambainis' algorithm has the following structure. Its state space is spanned by basic states $|T\rangle$, for all sets of indices $T \subseteq \{1, \ldots, N\}$ with $|T| = r$. The algorithm starts in a uniform superposition of all $|T\rangle$ and repeatedly applies a sequence of two transformations:

1. Conditional phase flip: $|T\rangle \to -|T\rangle$ for all T such that T contains i, j with $x_i = x_j$ and $|T\rangle \to |T\rangle$ for all other T;
2. Quantum walk: perform $O(\sqrt{r})$ steps of quantum walk, as defined in [2]. Each step is a transformation that maps each $|T\rangle$ to a combination of basis states $|T'\rangle$ for T' that differ from T in one element.

The algorithm maintains another quantum register, which stores all the values of x_i, $i \in T$. This register is updated with every step of the quantum walk.

If there are two elements i, j such that $x_i = x_j$, repeating these two transformations $O(N/r)$ times increases the amplitudes of $|T\rangle$ containing i, j. Measuring the state of the algorithm at that point with high probability produces a set T containing i, j. Then, from the set T, we can find i and j.

The basic structure of [2] is similar to Grover's quantum search, but with one substantial difference. In Grover's algorithm, instead of using a quantum walk, one would use Grover's *diffusion transformation*. Implementing Grover's diffusion requires $\Omega(r)$ updates to the register that stores x_i, $i \in T$. In contrast to Grover's diffusion, each step of quantum walk changes T by one element, requiring just one update to the list of x_i, $i \in T$. Thus, $O(\sqrt{r})$ steps of quantum walk can be performed with $O(\sqrt{r})$ updates, quadratically better than Grover's diffusion. And, as shown in [2], the quantum walk provides a sufficiently good approximation of diffusion for the algorithm to work correctly.

This was one of first uses of quantum walks to construct quantum algorithms. Ambainis, Kempe, Rivosh [4] then generalized it to handle searching on grids (described in another entry of this encyclopedia). Their algorithm is based on the same mathematical ideas, but has a slightly different structure. Instead of alternating quantum walk

```
1. Initialize $x$ to a state sampled from some initial distribution over the states of $P$.
2. $t_2$ times repeat:
   (a) If the current state $y$ is marked, output $y$ and stop;
   (b) Simulate $t_1$ steps of random walk, starting with the current state $y$.
3. If the algorithm has not terminated, output "no marked state".
```

Quantum Algorithm for Element Distinctness, Algorithm 1
Search by a classical random walk

steps with phase flips, it performs a quantum walk with two different walk rules – the normal walk rule and the "perturbed" one. (The normal rule corresponds to a walk without a phase flip and the "perturbed" rule corresponds to a combination of the walk with a phase flip).

Generalization to Arbitrary Markov Chains

Szegedy [14] and Magniez et al. [13] have generalized the algorithms of [4] and [2], respectively, to speed up the search of an arbitrary Markov chain. The main result of [13] is as follows.

Let P be an irreducible Markov chain with state space X. Assume that some states in the state space of P are *marked*. Our goal is to find a marked state. This can be done by a classical algorithm that runs the Markov chain P until it reaches a marked state (Algorithm 1).

There are 3 costs that contribute to the complexity of Algorithm 1:

1. **Setup cost** S**:** the cost to sample the initial state x from the initial distribution.
2. **Update cost** U**:** the cost to simulate one step of a random walk.
3. **Checking cost** C**:** the cost to check if the current state x is marked.

The overall complexity of the classical algorithm is then $S + t_2(t_1 U + C)$. The required t_1 and t_2 can be calculated from the characteristics of the Markov chain P. Namely,

Proposition 1 ([13]) *Let P be an ergodic, yet symmetric Markov chain. Let $\delta > 0$ be the eigenvalue gap of P and, assume that, whenever the set of marked states M is nonempty, we have $|M|/|X| \geq \epsilon$. Then there are $t_1 = O(1/\delta)$ and $t_2 = O(1/\epsilon)$ such that Algorithm 1 finds a marked element with high probability.*

Thus, the cost of finding a marked element classically is $O(S + 1/\epsilon(1/\delta U + C))$. Magniez et al. [13] construct a quantum algorithm that finds a marked element in

$O(S' + 1/\epsilon(1/\sqrt{\delta}U' + C'))$ steps, with S', U', C' being quantum versions of the setup, update and checking costs (in most of applications, these are of the same order as S, U and C). This achieves a quadratic improvement in the dependence on both ε and δ.

The element distinctness problem is solved by a particular case of this algorithm: a search on the Johnson graph. The Johnson graph is the graph whose vertices v_T correspond to subsets $T \subseteq \{1, \ldots, N\}$ of size $|T| = r$. A vertex v_T is connected to a vertex $v_{T'}$, if the subsets T and T' differ in exactly one element. A vertex v_T is *marked* if T contains indices i, j with $x_i = x_j$.

Consider the following Markov chain on the Johnson graph. The starting probability distribution s is the uniform distribution over the vertices of the Johnson graph. In each step, the Markov chain chooses the next vertex $v_{T'}$ from all vertices that are adjacent to the current vertex v_T, uniformly at random. While running the Markov chain, one maintains a list of all x_i, $i \in T$. This means that the costs of the classical Markov chain are as follows:

- Setup cost of $S = r$ queries (to query all x_i, $i \in T$ where v_T is the starting state).
- Update cost of $U = 1$ query (to query the value x_i, $i \in T' - T$, where v_T is the vertex before the step and $v_{T'}$ is the new vertex).
- Checking cost of $C = 0$ queries (the values x_i, $i \in T$ are already known to the algorithm, no further queries are needed).

The quantum costs S', U', C' are of the same order as S, U, C.

For this Markov chain, it can be shown that the eigenvalue gap is $\delta = O(1/r)$ and the fraction of marked states is $\epsilon = O((r^2)/(N^2))$. Thus, the quantum algorithm runs in time

$$O\left(S' + \frac{1}{\sqrt{\epsilon}}\left(\frac{1}{\sqrt{\delta}}U' + C'\right)\right) = O\left(S' + \sqrt{r}\left(\frac{N}{r}U' + C'\right)\right) = O\left(r + \frac{N}{\sqrt{r}}\right) .$$

Applications

Magniez et al. [12] showed how to use the ideas from the element distinctness algorithm as a subroutine to solve the *triangle problem*. In the triangle problem, one is given a graph G on n vertices, accessible by queries to an oracle, and they must determine whether the graph contains a triangle (three vertices v_1, v_2, v_3 with $v_1\,v_2$, $v_1\,v_3$ and $v_2\,v_3$ all being edges). This problem requires $\Omega(n^2)$ queries classically. Magniez et al. [12] showed that it can be solved using $O(n^{1.3}\log^c n)$ quantum queries, with a modification of the element distinctness algorithm as a subroutine. This was then improved to $O(n^{1.3})$ by [13].

The methods of Szegedy [14] and Magniez et al. [13] can be used as subroutines for quantum algorithms for checking matrix identities [7,11].

Open Problems

1. How many queries are necessary to solve the element distinctness problem if the memory accessible to the algorithm is limited to r items, $r < N^{2/3}$? The algorithm of [2] gives $O(N/\sqrt{r})$ queries, and the best lower bound is $\Omega(N^{2/3})$ queries.
2. Consider the following problem:
   Graph collision [12]. The problem is specified by a graph G (which is arbitrary but known in advance) and variables $x_1, \ldots, x_N \in \{0, 1\}$, accessible by queries to an oracle. The task is to determine if G contains an edge uv such that $x_u = x_v = 1$. How many queries are necessary to solve this problem?
   The element distinctness algorithm can be adapted to solve this problem with $O(N^{2/3})$ queries [12], but there is no matching lower bound. Is there a better algorithm? A better algorithm for the graph collision problem would immediately imply a better algorithm for the triangle problem.

Cross References

▶ Quantization of Markov Chains
▶ Quantum Algorithm for Finding Triangles
▶ Quantum Search

Recommended Reading

1. Aaronson, S., Shi, Y.: Quantum lower bounds for the collision and the element distinctness problems. J. ACM **51**(4), 595–605 (2004)
2. Ambainis, A.: Quantum walk algorithm for element distinctness. SIAM J. Comput. **37**(1), 210–239 (2007)
3. Ambainis, A.: Polynomial degree and lower bounds in quantum complexity: Collision and element distinctness with small range. Theor. Comput. **1**, 37–46 (2005)
4. Ambainis, A., Kempe, J., Rivosh, A.: In: Proceedings of the ACM/SIAM Symposium on Discrete Algorithms (SODA'06), 2006, pp. 1099–1108
5. Buhrman, H., Durr, C., Heiligman, M., Høyer, P., Magniez, F., Santha, M., de Wolf, R.: Quantum algorithms for element distinctness. SIAM J. Comput. **34**(6), 1324–1330 (2005)
6. Borodin, A., Fischer, M., Kirkpatrick, D., Lynch, N.: A time-space tradeoff for sorting on non-oblivious machines. J. Comput. Syst. Sci. **22**, 351–364 (1981)
7. Buhrman, H., Spalek, R.: Quantum verification of matrix products. In: Proceedings of the ACM/SIAM Symposium on Discrete Algorithms (SODA'06), 2006, pp. 880–889

8. Beame, P., Saks, M., Sun, X., Vee, E.: Time-space trade-off lower bounds for randomized computation of decision problems. J. ACM **50**(2), 154–195 (2003)
9. Childs, A.M., Eisenberg, J.M.: Quantum algorithms for subset finding. Quantum Inf. Comput. **5**, 593 (2005)
10. Kutin, S.: Quantum lower bound for the collision problem with small range. Theor. Comput. **1**, 29–36 (2005)
11. Magniez, F., Nayak, A.: Quantum complexity of testing group commutativity. In: Proceedings of the International Colloquium Automata, Languages and Programming (ICALP'05), 2005, pp. 1312–1324
12. Magniez, F., Santha, M., Szegedy, M.: Quantum algorithms for the triangle problem. SIAM J. Comput. **37**(2), 413–424 (2007)
13. Magniez, F., Nayak, A., Roland, J., Santha, M.: Search by quantum walk. In: Proceedings of the ACM Symposium on the Theory of Computing (STOC'07), 2007, pp. 575–584
14. Szegedy, M.: Quantum speed-up of Markov Chain based algorithms. In: Proceedings of the IEEE Conference on Foundations of Computer Science (FOCS'04), 2004, pp. 32–41
15. Yao, A.: Near-optimal time-space tradeoff for element distinctness. SIAM J. Comput. **23**(5), 966–975 (1994)

Quantum Algorithm for Factoring

1994; Shor

SEAN HALLGREN
Department of Computer Science and Engineering, The Pennsylvania State University, University Park, PA, USA

Problem Definition

Every positive integer n has a unique decomposition as a product of primes $n = p_1^{e_1} \cdots p_k^{e_k}$, for primes numbers p_i and positive integer exponents e_i. Computing the decomposition $p_1, e_1, \ldots, p_k, e_k$ from n is the factoring problem.

Factoring has been studied for many hundreds of years and exponential time algorithms for it were found that include trial division, Lehman's method, Pollard's ρ method, and Shank's class group method [1]. With the invention of the RSA public-key cryptosystem in the late 1970s, the problem became practically important and started receiving much more attention. The security of RSA is closely related to the complexity of factoring, and in particular, it is only secure if factoring does not have an efficient algorithm. The first subexponential-time algorithm is due to Morrison and Brillhard [4] using a continued fraction algorithm. This was succeeded by the quadratic sieve method of Pomerance and the elliptic curve method of Lenstra [5]. The Number Field Sieve [2,3], found in 1989, is the best known classical algorithm for factoring and runs in time $\exp(c(\log n)^{1/3}(\log\log n)^{2/3})$ for some constant c. Shor's result is a polynomial-time quantum algorithm for factoring.

Key Results

Theorem 1 ([2,3]) *There is a subexponential-time classical algorithm that factors the integer n in time $\exp(c(\log n)^{1/3}(\log\log n)^{2/3})$.*

Theorem 2 ([6]) *There is a polynomial-time quantum algorithm that factors integers. The algorithm factors n in time $O((\log n)^2(\log n \log n)(\log\log\log n))$ plus polynomial in $\log n$ post-processing which can be done classically.*

Applications

Computationally hard number theoretic problems are useful for public key cryptosystems. The RSA public-key cryptosystem, as well as others, require that factoring not have an efficient algorithm. The best known classical algorithms for factoring can help determine how secure the cryptosystem is and what key sizes to choose. Shor's quantum algorithm for factoring can break these systems in polynomial-time using a quantum computer.

Open Problems

Whether there is a polynomial-time classical algorithm for factoring is open. There are problems which are harder than factoring such as finding the unit group of an arbitrary degree number field for which no efficient quantum algorithm has been found yet.

Cross References

► Quantum Algorithm for the Discrete Logarithm Problem
► Quantum Algorithms for Class Group of a Number Field
► Quantum Algorithm for Solving the Pell's Equation

Recommended Reading

1. Cohen, H.: A course in computational algebraic number theory. Graduate Texts in Mathematics, vol. 138. Springer (1993)
2. Lenstra, A., Lenstra, H. (eds.): The Development of the Number Field Sieve. Lecture Notes in Mathematics, vol. 1544. Springer (1993)
3. Lenstra, A.K., Lenstra, H.W. Jr., Manasse, M.S., Pollard, J.M.: The number field sieve. In: Proceedings of the Twenty Second Annual ACM Symposium on Theory of Computing, Baltimore, Maryland, 14–16 May 1990, pp. 564–572
4. Morrison, M., Brillhart, J.: A method of factoring and the factorization of F_7
5. Pomerance, C.: Factoring. In: Pomerance, C. (ed.) Cryptology and Computational Number Theory, Proceedings of Symposia in Applied Mathematics, vol. 42, p. 27. American Mathematical Society
6. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM J. Comput. **26**, 1484–1509 (1997)

Quantum Algorithm for Finding Triangles

2005; Magniez, Santha, Szegedy

PETER RICHTER
Department of Computer Science, Rutgers, The State University of New Jersey, Piscataway, NJ, USA

Keywords and Synonyms

Triangle finding

Problem Definition

A *triangle* is a clique of size three in an undirected graph. Triangle finding is a fundamental computational problem whose time complexity is closely related to that of matrix multiplication. It has been the subject of considerable study recently as a basic search problem whose quantum query complexity is still unclear, in contrast to the unstructured search problem [4,10] and the element distinctness problem [1,3]. This survey concerns quantum query algorithms for triangle finding.

Notation and Constraints

A *quantum query algorithm* $Q_f : |\psi_0\rangle \mapsto |\psi_f\rangle$ computes a property P of a function f by mapping the initial state $|\psi_0\rangle = |0\rangle|0\rangle|0\rangle$ (in which its *query*, *answer*, and *workspace* registers are cleared) to a final state $|\psi_f\rangle = Q_f|\psi_0\rangle$ by applying a sequence $Q_f = U_k O_f U_{k-1} O_f \cdots U_1 O_f U_0$ of unitary operators on the complex vector space spanned by all possible basis states $|x\rangle|a\rangle|z\rangle$. The unitary operators are of two types: *oracle queries* $O_f : |x\rangle|a\rangle|z\rangle \mapsto |x\rangle|a \oplus f(x)\rangle|z\rangle$, which yield information about f, and *non-query steps* U_k, which are independent of f. The *quantum query complexity* of P is the minimum number of oracle queries required by a quantum query algorithm computing P with probability at least 2/3.

Consider the *triangle finding problem* for an unknown (simple, undirected) graph $G \subseteq \{(a,b) : a, b \in [n], a \neq b\}$ on vertices $[n] = \{1, \dots, n\}$ and $m = |G|$ undirected edges, where $(a,b) = (b,a)$ by convention. The function f to query is the adjacency matrix of G and the property P to be computed is whether or not G contains a triangle.

Problem 1 (Triangle finding)
INPUT: *The adjacency matrix f of a graph G on n vertices.*
OUTPUT: *A triangle with probability $\geq 2/3$ if one exists (search version), or a boolean value indicating whether or not one exists with probability $\geq 2/3$ (decision version).*

A lower bound of $\Omega(n)$ on the quantum query complexity of the triangle finding problem was shown by Buhrman et al. [6]. The trivial upper bound of $O(n^2)$ is attainable by querying every entry of f classically.

Classical Results

The classical randomized query complexity of a problem is defined similarly to the quantum query complexity, only the operators U_k are stochastic rather than unitary; in particular, this means oracle queries can be made according to classical distributions but not quantum superpositions. It is easy to see that the randomized query complexity of the triangle finding problem (search and decision versions) is $\Theta(n^2)$.

Key Results

Improvement of the upper bound on the quantum query complexity of the triangle finding problem has stemmed from two lines of approach: increasingly clever utilization of structure in the search space (combined with standard quantum amplitude amplification) and application of quantum walk search procedures.

An $O(n + \sqrt{nm})$ Algorithm Using Amplitude Amplification

Since there are $\binom{n}{3}$ potential triangles (a,b,c) in G, a trivial application of Grover's quantum search algorithm [10] solves the triangle finding problem with $O(n^{3/2})$ quantum queries. Buhrman et al. [6] improved this upper bound in the special case where G is *sparse* (i. e., $m = o(n^2)$) by the following argument.

Suppose Grover's algorithm is used to find (a) an edge $(a,b) \in G$ among all $\binom{n}{2}$ potential edges, followed by (b) a vertex $c \in [n]$ such that (a,b,c) is a triangle in G. The costs of steps (a) and (b) are $O(\sqrt{n^2/m})$ and $O(\sqrt{n})$ quantum queries, respectively. If G contains a triangle Δ, then step (a) will find an edge (a,b) from Δ with probability $\Omega(1/m)$, and step (b) will find the third vertex c in the triangle $\Delta = (a,b,c)$ with constant probability. Therefore, steps (a) and (b) together find a triangle with probability $\Omega(1/m)$. By repeating the steps $O(\sqrt{m})$ times using amplitude amplification (Brassard et al. [5]), one can find a triangle with probability 2/3. The total cost is $O(\sqrt{m}(\sqrt{n^2/m} + \sqrt{n})) = O(n + \sqrt{nm})$ quantum queries. Summarizing:

Theorem 1 (Buhrman et al. [6]) *Using quantum amplitude amplification, the triangle finding problem can be solved in $O(n + \sqrt{nm})$ quantum queries.*

An $\tilde{O}(n^{10/7})$ Algorithm Using Amplitude Amplification

Let μ^2 be the complete graph on vertices $\mu \subseteq [n]$, $\nu_G(v)$ be the set of vertices adjacent to a vertex v, and $deg_G(v)$ be the degree of v. Note that for any vertex $v \in [n]$, one can either find a triangle in G containing v or verify that $G \subseteq [n]^2 \setminus \nu_G(v)^2$ with $\tilde{O}(n)$ quantum queries and success probability $1 - 1/n^3$, by first computing $\nu_G(v)$ classically and then using Grover's search logarithmically many times to find an edge of G in $\nu_G(v)^2$ with high probability if one exists. Szegedy et al [13,14]. use this observation to design an algorithm for the triangle finding problem that utilizes no quantum procedure other than amplitude amplification (just like the algorithm of Buhrman et al [6].) yet requires only $\tilde{O}(n^{10/7})$ quantum queries.

The algorithm of Szegedy et al. [13,14]. is as follows. First, select $k = \tilde{O}(n^{\epsilon})$ vertices $v_1, \ldots, v_k$ uniformly at random from $[n]$ without replacement and compute each $\nu_G(v_i)$. At a cost of $\tilde{O}(n^{1+\epsilon})$ quantum queries, one can either find a triangle in G containing one of the v_i or conclude with high probability that $G \subseteq G' := [n]^2 \setminus \cup_i \nu_G(v_i)^2$. Suppose the latter. Then it can be shown that with high probability, one can construct a partition (T, E) of G' such that T contains $O(n^{3-\epsilon'})$ triangles and $E \cap G$ has size $O(n^{2-\delta} + n^{2-\epsilon+\delta+\epsilon'})$ in $\tilde{O}(n^{1+\delta+\epsilon'})$ queries (or one will find a triangle in G in the process). Since $G \subseteq G'$, every triangle in G either lies within T or intersects E. In the first case, one will find a triangle in $G \cap T$ in $O(\sqrt{n^{3-\epsilon'}})$ quantum queries by searching G with Grover's algorithm for a triangle in T, which is known from the partitioning procedure. In the second case, one will find a triangle in G with an edge in E in $\tilde{O}(n + \sqrt{n^{3-\min\{\delta,\epsilon-\delta-\epsilon'\}}})$ quantum queries using the algorithm of Buhrman et al.[6] with $m = |G \cap E|$. Thus:

Theorem 2 (Szegedy et al. [13,14]) *Using quantum amplitude amplification, the triangle finding problem can be solved in $\tilde{O}(n^{1+\epsilon} + n^{1+\delta+\epsilon'} + \sqrt{n^{3-\epsilon'}} + \sqrt{n^{3-\min\{\delta,\epsilon-\delta-\epsilon'\}}})$ quantum queries.*

Letting $\epsilon = 3/7$ and $\epsilon' = \delta = 1/7$ yields an algorithm using $\tilde{O}(n^{10/7})$ quantum queries.

An $\tilde{O}(n^{13/10})$ Algorithm Using Quantum Walks

A more efficient algorithm for the triangle finding problem was obtained by Magniez et al. [13], using the quantum walk search procedure introduced by Ambainis [3] to obtain an optimal quantum query algorithm for the element distinctness problem.

Given oracle access to a function f defining a relation $C \subseteq [n]^k$ Ambainis' search procedure solves the *k-collision problem*: find a pair $(a_1, \ldots, a_k) \in C$ if one exists. The search procedure operates on three quantum registers $|A\rangle|D(A)\rangle|y\rangle$: the *set* register $|A\rangle$ holds a set $A \subseteq [n]$ of size $|A| = r$, the *data* register $|D(A)\rangle$ holds a data structure $D(A)$, and the *coin* register $|y\rangle$ holds an element $i \notin A$. By checking the data structure $D(A)$ using a quantum query procedure Φ with *checking cost* $c(r)$, one can determine whether or not $A^k \cap C \neq \emptyset$. Suppose $D(A)$ can be constructed from scratch at a *setup cost* $s(r)$ and modified from $D(A)$ to $D(A')$ where $|A \cap A'| = r - 1$ at an *update cost* $u(r)$. Then Ambainis' quantum walk search procedure solves the k-collision problem in $\tilde{O}(s(r) + (\frac{n}{r})^{k/2} \cdot (c(r) + \sqrt{r} \cdot u(r)))$ quantum queries. (For details, see the encyclopedia entry on element distinctness.)

Consider the *graph collision* problem on a graph $G \subseteq [n]^2$, where f defines the binary relation $C \subseteq [n]^2$ satisfying $C(u, u')$ if $f(u) = f(u') = 1$ and $(u, u') \in G$. Ambainis' search procedure solves the graph collision problem in $\tilde{O}(n^{2/3})$ quantum queries, by the following argument. Fix $k = 2$ and $r = n^{2/3}$ in the k-collision algorithm, and for every $U \subseteq [n]$ define $D(U) = \{(v, f(v)) : v \in U\}$ and $\Phi(D(U)) = 1$ if some $u, u' \in U$ satisfies C. Then $s(r) = r$ initial queries $f(v)$ are needed to set up $D(U)$, $u(r) = 1$ new query $f(v)$ is needed to update $D(U)$, and $c(r) = 0$ additional queries $f(v)$ are needed to check $\Phi(D(U))$. Therefore, $\tilde{O}(r + \frac{n}{r}(\sqrt{r})) = \tilde{O}(n^{2/3})$ queries are needed altogether.

Magniez et al. [13] solve the triangle finding problem by reduction to the graph collision problem. Again fix $k = 2$ and $r = n^{2/3}$. Let C be the set of edges contained in at least one triangle. Define $D(U) = G|_U$ and $\Phi(D(U)) = 1$ if some edge in $G|_U$ satisfies C. Then $s(r) = O(r^2)$ initial queries are needed to set up $D(U)$ and $u(r) = r$ new queries are needed to update $D(U)$. It remains to bound the checking cost $c(r)$. For any vertex $v \in [n]$, consider the graph collision oracle f_v on $G|_U$ satisfying $f_v(u) = 1$ if $(u, v) \in G$. An edge of $G|_U$ is a triangle in G if and only if the edge is a solution to the graph collision problem on $G|_U$ for some $v \in [n]$. This problem can be solved for a particular v in $\tilde{O}(r^{2/3})$ queries. Using $\tilde{O}(\sqrt{n})$ steps of amplitude amplification, one can find out if *any* $v \in [n]$ generates an accepting solution to the graph collision problem with high probability. Hence, the checking cost is $c(r) = \tilde{O}(\sqrt{n} \cdot r^{2/3})$ queries, from which it follows that:

Theorem 3 (Magniez et al. [13]) *Using a quantum walk search procedure, the triangle finding problem can be solved in $\tilde{O}(r^2 + \frac{n}{r}(\sqrt{n} \cdot r^{2/3} + \sqrt{r} \cdot r))$ quantum queries.*

Letting $r = n^{3/5}$ yields an algorithm using $\tilde{O}(n^{13/10})$ quantum queries.

In recent work Magniez et al. [12], using the quantum walk defined by Szegedy [15], have introduced a new quantum walk search procedure generalizing that of Ambainis [3]. Among the consequences is a quantum walk algorithm for triangle finding in $O(n^{13/10})$ quantum queries.

Applications

Extensions of the quantum walk algorithm for triangle finding have been used to find cliques and other fixed subgraphs in a graph and to decide monotone graph properties with small certificates using fewer quantum queries than previous algorithms.

Finding Cliques, Subgraphs, and Subsets

Ambainis' k-collision algorithm [3] can find a copy of any graph H with $k > 3$ vertices in $\tilde{O}(n^{2-2/(k+1)})$ quantum queries. In the case where H is a k-clique, Childs and Eisenberg [9] gave an $\tilde{O}(n^{2.5-6/(k+2)})$ query algorithm. A simple generalization of the triangle finding quantum walk algorithm of Magniez et al. [13] improves this to $\tilde{O}(n^{2-2/k})$.

Monotone Graph Properties

Recall that a *monotone graph property* is a boolean property of a graph whose value is invariant under permutation of the vertex labels and monotone under any sequence of edge deletions. Examples of monotone graph properties are connectedness, planarity, and triangle-freeness. A *1-certificate* is a minimal subset of edge queries proving that a property holds (e. g., three edges suffice to prove that a graph contains a triangle). Magniez et al. [13] show that their quantum walk algorithm for the triangle finding problem can be generalized to an $\tilde{O}(n^{2-2/k})$ quantum query algorithm deciding any monotone graph property with 1-certificates of size at most $k > 3$ vertices. The best known lower bound is $\Omega(n)$.

Open Problems

The most obvious remaining open problem is to resolve the quantum query complexity of the triangle finding problem; again, the best upper and lower bounds currently known are $O(n^{13/10})$ and $\Omega(n)$. Beyond this, there are the following open problems:

Quantum Query Complexity of Monotone Graph Properties

The best known lower bound for the quantum query complexity of (nontrivial) monotone graph properties is $\Omega(n^{2/3} \log^{1/6} n)$, observed by Andrew Yao to follow from the classical randomized lower bound $\Omega(n^{4/3} \log^{1/3} n)$ of Chakrabarti and Khot [8] and the quantum adversary technique of Ambainis [2]. Is an improvement to $\Omega(n)$ possible? If so, this would be tight, since one can determine whether the edge set of a graph is nonempty in $O(n)$ quantum queries using Grover's algorithm.

New Quantum Walk Algorithms

Quantum walks have been successfully applied in designing more efficient quantum search algorithms for several problems, including element distinctness [3], triangle finding [13], matrix product verification [7], and group commutativity testing [11]. It would be nice to see how far the quantum walk approach can be extended to obtain new and better quantum algorithms for various computational problems.

Cross References

► Quantization of Markov Chains
► Quantum Algorithm for Element Distinctness

Recommended Reading

1. Aaronson, S., Shi, Y.: Quantum lower bounds for the collision and the element distinctness problems. J. ACM **51**(4), 595–605, (2004), quant-ph/0112086
2. Ambainis, A.: Quantum lower bounds by quantum arguments. J. Comput. Syst. Sci. **64**, 750–767, (2002), quant-ph/0002066
3. Ambainis, A.: Quantum walk algorithm for element distinctness. SIAM J. Comput. **37**(1), 210–239, (2007) Preliminary version in Proc. FOCS, (2004), quant-ph/0311001
4. Bennett, C., Bernstein, E., Brassard, G., Vazirani, U.: Strengths and weaknesses of quantum computing. SIAM J. Comput. **26**(5), 1510–1523, (1997), quant-ph/9701001
5. Brassard, G., Høyer, P., Mosca, M., Tapp, A.: Quantum amplitude amplification and estimation. In: Quantum Computation and Quantum Information: A Millennium Volume, AMS Contemporary Mathematics Series, vol. 305. (2002) quant-ph/0005055
6. Buhrman, H., Dürr, C., Heiligman, M., P.Høyer, Magniez, F., Santha, M., de Wolf, R.: Quantum algorithms for element distinctness. SIAM J. Computing **34**(6), 1324–1330, (2005). Preliminary version in Proc. CCC (2001) quant-ph/0007016
7. Buhrman, H., Spalek, R.: Quantum verification of matrix products. Proc. SODA, (2006) quant-ph/0409035
8. Chakrabarti, A., Khot, S.: Improved lower bounds on the randomized complexity of graph properties. Proc. ICALP (2001)
9. Childs, A., Eisenberg, J.: Quantum algorithms for subset finding. Quantum Inf. Comput. **5**, 593 (2005), quant-ph/0311038
10. Grover, L.: A fast quantum mechanical algorithm for database search. Proc. STOC (1996) quant-ph/9605043
11. Magniez, F., Nayak, A.: Quantum complexity of testing group commutativity. Algorithmica **48**(3), 221–232 (2007) Preliminary version in Proc. ICALP (2005) quant-ph/0506265

12. Magniez, F., Nayak, A., Roland, J., Santha, M.: Search via quantum walk. Proc. STOC (2007) quant-ph/0608026
13. Magniez, F., Santha, M., Szegedy, M.: Quantum algorithms for the triangle problem. SIAM J. Comput. **37**(2), 413–424, (2007). Preliminary version in Proc. SODA (2005) quant-ph/0310134
14. Szegedy, M.: On the quantum query complexity of detecting triangles in graphs. quant-ph/0310107
15. Szegedy, M.: Quantum speed-up of Markov chain based algorithms. Proc. FOCS (2004) quant-ph/0401053

Quantum Algorithm for the Parity Problem

1985; Deutsch

YAOYUN SHI
Department of Electrical Engineering and Computer Science, University of Michigan,
Ann Arbor, MI, USA

Keywords and Synonyms

Parity; Deutsch–Jozsa algorithm; Deutsch algorithm

Problem Definition

The *parity* of n bits $x_0, x_1, \cdots, x_{n-1} \in \{0, 1\}$ is

$$x_0 \oplus x_1 \oplus \cdots \oplus x_{n-1} = \sum_{i=0}^{n-1} x_i \mod 2 .$$

As an elementary Boolean function, Parity is important not only as a building block of digital logic, but also for its instrumental roles in several areas such as error-correction, hashing, discrete Fourier analysis, pseudorandomness, communication complexity, and circuit complexity. The feature of Parity that underlies its many applications is its maximum sensitivity to the input: flipping any bit in the input changes the output. The computation of Parity from its input bits is quite straightforward in most computation models. However, two settings deserve attention.

The first is the circuit complexity of Parity when the gates are restricted to AND, OR, and NOT gates. It is known that Parity cannot be computed by such a circuit of a polynomial size and a constant depth, a groundbreaking result proved independently by Furst, Saxe, and Sipser [7], and Ajtai [1], and improved by several subsequent works.

The second, and the focus of this article, is in the decision tree model (also called the query model or the black-box model), where the input bits $x = x_0 x_1 \cdots x_{n-1} \in \{0, 1\}^n$ are known to an oracle only, and the algorithm needs to ask questions of the type "$x_i = ?$" to access the input. The complexity is measured by the number of queries. Specifically, a quantum query is the application of the following query gate

$$O_x : |i, b\rangle \mapsto |i, b \oplus x_i\rangle , \quad i \in \{0, \cdots, n-1\}, b \in \{0, 1\} .$$

Key Results

Proposition 1 *There is a quantum query algorithm computing the parity of 2 bits with probability 1 using 1 query.*

Proof Denote by $|\pm\rangle = \frac{1}{\sqrt{2}}(|0\rangle \pm |1\rangle)$. The initial state of the algorithm is

$$\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes |-\rangle .$$

Apply a query gate, using the first register for the index slot and the second register for the answer slot. The resulting state is

$$\frac{1}{\sqrt{2}}((-1)^{x_0}|0\rangle + (-1)^{x_1}|1\rangle) \otimes |-\rangle .$$

Applying a Hadamard gate $H = |+\rangle\langle 0| + |-\rangle\langle 1|$ on the first register brings the state to

$$(-1)^{x_0}|x_0 + x_1\rangle \otimes |-\rangle .$$

Thus measuring the first register gives $x_0 + x_1$ with certainty. □

Corollary 2 *There is a quantum query algorithm computing the parity of n bits with probability 1 using $\lceil n/2 \rceil$ queries.*

The above quantum upper bound for Parity is tight, even if the algorithm is allowed to err with a probability bounded away from 1/2 [6]. In contrast, any classical randomized algorithm with bounded error probability requires n queries. This follows from the fact that on a random input, any classical algorithm not knowing all the input bits is correct with precisely 1/2 probability.

Applications

The quantum speedup for computing Parity was first observed by Deutsch [4]. His algorithm uses $|0\rangle$ in the answer slot, instead of $|-\rangle$. After one query, the algorithm has 3/4 chance of computing the parity, better than any classical algorithm (1/2 chance). The presented algorithm is actually a special case of the Deutsch–Jozsa Algorithm, which solves the following problem now referred to as the Deutsch–Jozsa Problem.

Problem 1 (Deutsch–Jozsa Problem) *Let $n \geq 1$ be an integer. Given an oracle function $f\colon \{0,1\}^n \to \{0,1\}$ that satisfies either (a) $f(x)$ is constant on all $x \in \{0,1\}^n$, or (b) $|\{x\colon f(x)=1\}| = |\{x\colon f(x)=0\}| = 2^{n-1}$, determine which case it is.*

When $n = 1$, the above problem is precisely Parity of 2 bits. For a general n, the Deutsch–Jozsa Algorithm solves the problem using only once the following query gate

$$O_f\colon |x,b\rangle \mapsto |x, f(x) \oplus b\rangle\,, \quad x \in \{0,1\}^n,\ b \in \{0,1\}\,.$$

The algorithm starts with

$$|0^n\rangle \otimes |-\rangle\,.$$

It applies $H^{\otimes n}$ on the index register (the first n qubits), changing the state to

$$\frac{1}{2^{n/2}} \sum_{x \in \{0,1\}^n} |x\rangle \otimes |-\rangle\,.$$

The oracle gate is then applied, resulting in

$$\frac{1}{2^{n/2}} \sum_{x \in \{0,1\}^n} (-1)^{f(x)}|x\rangle \otimes |-\rangle\,.$$

For the second time, $H^{\otimes n}$ is applied on the index register, bringing the state to

$$\sum_{y \in \{0,1\}^n} \left(\frac{1}{2^n} \sum_{x \in \{0,1\}^n} (-1)^{f(x)+x\cdot y} \right) |y\rangle \otimes |-\rangle\,. \qquad (1)$$

Finally, the index register is measured in the computational basis. The Algorithm returns "Case (a)" if 0^n is observed, otherwise returns "Case (b)".

By direct inspection, the amplitude of $|0^n\rangle$ is 1 in Case (a), and 0 in Case (b). Thus the Algorithm is correct with probability 1. It is easy to see that any deterministic algorithm requires $n/2 + 1$ queries in the worst case, thus the Algorithm provides the first exponential quantum versus deterministic speedup.

Note that $O(1)$ expected number of queries are sufficient for randomized algorithms to solve the Deutsch–Jozsa Problem with a constant success probability arbitrarily close to 1. Thus the Deutsch–Jozsa Algorithm does not have much advantage compared with error-bounded randomized algorithms. One might also feel that the saving of one query for computing the parity of 2 bits by Deutsch–Jozsa Algorithm is due to the artificial definition of one quantum query. Thus the significance of the Deutsch–Jozsa Algorithm is not in solving a practical problem, but in its pioneering use of Quantum Fourier Transform (QFT), of which $H^{\otimes n}$ is one, in the pattern

$$\text{QFT} \to \text{Query} \to \text{QFT}\,.$$

The same pattern appears in many subsequent quantum algorithms, including those found by Bernstein and Vazirani [2], Simon [8], Shor.

The Deutsch–Jozsa Algorithm is also referred to as Deutsch Algorithm. The Algorithm as presented above is actually the result of the improvement by Cleve, Ekert, Macchiavello, and Mosca [3] and independently by Tapp (unpublished) on the algorithm in [5].

Cross References

► Greedy Set-Cover Algorithms

Recommended Reading

1. Ajtai, M.: $\sum_1^1$-formulae on finite structures. Ann. Pure Appl. Log. **24**(1), 1–48 (1983)
2. Bernstein, E., Vazirani, U.: Quantum complexity theory. SIAM J. Comput. **26**(5), 1411–1473 (1997)
3. Cleve, R., Ekert, A., Macchiavello, C., Mosca, M.: Quantum algorithms revisited. Proc. Royal Soc. London **A454**, 339–354 (1998)
4. Deutsch, D.: Quantum theory, the Church-Turing principle and the universal quantum computer. Proc. Royal Soc. London **A400**, 97–117 (1985)
5. Deutsch, D., Jozsa, R.: Rapid solution of problems by quantum computation. Proc. Royal Soc. London **A439**, 553–558 (1992)
6. Farhi, E., Goldstone, J., Gutmann, S., Sipser, M.: A limit on the speed of quantum computation in determining parity. Phys. Rev. Lett. **81**, 5442–5444 (1998)
7. Furst, M., Saxe, J., Sipser, M.: Parity, circuits, and the polynomial time hierarchy. Math. Syst. Theor. **17**(1), 13–27 (1984)
8. Simon, D.R.: On the power of quantum computation. SIAM J. Comput. **26**(5), 1474–1483 (1997)

Quantum Algorithms for Class Group of a Number Field

2005; Hallgren

SEAN HALLGREN
Department of Computer Science and Engineering, The Pennsylvania State University, University Park, PA, USA

Problem Definition

Associated with each number field is a finite abelian group called the class group. The order of the class group is called the class number. Computing the class number and the structure of the class group of a number field are among the main tasks in computational algebraic number theory [3].

A number field F can be defined as a subfield of the complex numbers $\mathbb{C}$ which is generated over the rational numbers $\mathbb{Q}$ by an algebraic number, i. e. $F = \mathbb{Q}(\theta)$ where θ is the root of a polynomial with rational coefficients. The ring of integers $\mathcal{O}$ of F is the subset consisting of all elements that are roots of monic polynomials with integer coefficients. The ring $\mathcal{O} \subseteq F$ can be thought of as a generalization of $\mathbb{Z}$, the ring of integers in $\mathbb{Q}$. In particular, one can ask whether $\mathcal{O}$ is a principal ideal domain and whether elements in $\mathcal{O}$ have unique factorization. Another interesting problem is computing the unit group $\mathcal{O}^*$, which is the set of invertible algebraic integers inside F, that is, elements $\alpha \in \mathcal{O}$ such that α^{-1} is also in $\mathcal{O}$.

Ever since the class group was discovered by Gauss in 1798 it has been an interesting object of study. The class group of F is the set of equivalence classes of fractional ideals of F, where two ideals I and J are equivalent if there exists $\alpha \in F^*$ such that $J = \alpha I$. Multiplication of two ideals I and J is defined as the ideal generated by all products ab, where $a \in I$ and $b \in J$. Much is still unknown about number fields, such as whether there exist infinitely many number fields with trivial class group. The question of the class group being trivial is equivalent to asking whether the elements in the ring of integers $\mathcal{O}$ of the number field have unique factorization.

In addition to computing the class number and the structure of the class group, computing the unit group and determining whether given ideals are principal, called the principal ideal problem, are also central problems in computational algebraic number theory.

Key Results

The best known classical algorithms for the class group take subexponential time [1,3]. Assuming the GRH, computing the class group, the unit group, and solving the principal ideal problem are in NP∩CoNP [7].

The following theorems state that the three problems defined above have efficient quantum algorithms [4,6].

Theorem 1 *There is a polynomial-time quantum algorithm that computes the unit group of a constant degree number field.*

Theorem 2 *There is a polynomial-time quantum algorithm that solves the principal ideal problem in constant degree number fields.*

Theorem 3 *The class group and class number of a constant degree number field can be computed in quantum polynomial-time assuming the GRH.*

Computing the class group means computing the structure of a finite abelian group given a set of generators for it. When it is possible to efficiently multiply group elements and efficiently compute unique representations of each group element, then this problem reduces to the standard hidden subgroup problem over the integers, and therefore has an efficient quantum algorithm. Ideal multiplication is efficient in number fields. For imaginary number fields, there are efficient classical algorithms for computing group elements with a unique representation, and therefore there is an efficient quantum algorithm for computing the class group.

For real number fields, there is no known way to efficiently compute unique representations of class group elements. As a result, the classical algorithms typically compute the unit group and class group at the same time. A quantum algorithm [4] is able to efficiently compute the unit group of a number field, and then use the principal ideal algorithm to compute a unique quantum representation of each class group element. Then the standard quantum algorithm can be applied to compute the class group structure and class number.

Applications

There are factoring algorithms based on computing the class group of an imaginary number fields. One is exponential time and the other is subexponential-time [3].

Computationally hard number theoretic problems are useful for public key cryptosystems. Pell's equation reduces to the principal ideal problem, which forms the basis of the Buchmann-Williams key-exchange protocol [2]. Identification schemes have also been based on this problem by Hamdy and Maurer [5]. The classical exponential-time algorithms help determine which parameters to choose for the cryptosystem. Factoring reduces to Pell's equation and the best known algorithm for it is exponentially slower than the best factoring algorithm. Systems based on these harder problems were proposed as alternatives in case factoring turns out to be polynomial-time solvable. The efficient quantum algorithms can break these cryptosystems.

Open Problems

It remains open whether these problems can be solved in arbitrary degree number fields. The solution for the unit group can be thought of in terms of the hidden subgroup problem. That is, there exists a function on $\mathbb{R}^c$ which is constant on values that differ by an element of the unit lattice, and is one-to-one within the fundamental parallelepiped. However, this function cannot be evaluated efficiently since it has an uncountable domain, and instead an efficiently computable approximation must be used. To evaluate this discrete version of the function, a classical algorithm is used to compute reduced ideals near a given point in $\mathbb{R}^c$. This algorithm is only polynomial-time for constant degree number fields as it computes the shortest vector in a lattice. Such an algorithm can be used to set up a superposition over points approximating the points in the a coset of the unit lattice. After setting up the superposition, it must be shown that Fourier sampling, i. e. computing the Fourier transform and measuring, suffices to compute the lattice.

Cross References

► Quantum Algorithm for Factoring
► Quantum Algorithm for Solving the Pell's Equation

Recommended Reading

1. Buchmann, J.: A subexponential algorithm for the determination of class groups and regulators of algebraic number fields. In: Goldstein, C. (ed.) Séminaire de Théorie des Nombres, Paris 1988–1989, Progress in Mathematics, vol. 91, pp. 27–41. Birkhäuser (1990)
2. Buchmann, J.A., Williams, H.C.: A key exchange system based on real quadratic fields (extended abstract). In: Brassard, G. (ed.) Advances in Cryptology–CRYPTO '89. Lecture Notes in Computer Science, vol. 435, 20–24 Aug 1989, pp. 335–343. Springer (1990)
3. Cohen, H., A course in computational algebraic number theory, vol. 138 of Graduate Texts in Mathematics. Springer (1993)
4. Hallgren, S.: Fast quantum algorithms for computing the unit group and class group of a number field. In: Proceedings of the 37th ACM Symposium on Theory of Computing. (2005)
5. Hamdy, S., Maurer, M.: Feige-fiat-shamir identification based on real quadratic fields, Tech. Report TI-23/99. Technische Universität Darmstadt, Fachbereich Informatik. http://www.informatik.tu-darmstadt.de/TI/Veroeffentlichung/TR/ (1999)
6. Schmidt, A., Vollmer, U.: Polynomial time quantum algorithm for the computation of the unit group of a number field. In: Proceedings of the 37th ACM Symposium on Theory of Computing. (2005)
7. Thiel, C.: On the complexity of some problems in algorithmic algebraic number theory, Ph. D. thesis. Universität des Saarlandes, Saarbrücken, Germany (1995)

Quantum Algorithm for Search on Grids

2005; Ambainis, Kempe, Rivosh

ANDRIS AMBAINIS
Department of Computer Science, University of Latvia, Riga, Latvia

Keywords and Synonyms

Spatial search

Problem Definition

Consider an $\sqrt{N} \times \sqrt{N}$ grid, with each location storing a bit that is 0 or 1. The locations on the grid are indexed by (i, j), where $i, j \in \{0, 1, \ldots, \sqrt{N} - 1\}.a_{i,j}$ denotes the value stored at the location (i, j).

The task is to find a location storing $a_{i,j} = 1$. This problem is as an abstract model for search in a two-dimensional database, with each location storing a variable $x_{i,j}$ with more than two values. The goal is to find $x_{i,j}$ that satisfies certain constraints. One can then define new variables $a_{i,j}$ with $a_{i,j} = 1$ if $x_{i,j}$ satisfies the constraints and search for i, j satisfying $a_{i,j} = 1$.

The grid is searched by a "robot", which, at any moment of time is at one location i, j. In one time unit, the robot can either examine the current location or move one step in one of four directions (left, right, up or down).

In a probabilistic version of this model, the robot is probabilistic. It makes its decisions (querying the current location or moving) randomly according to pre-specified probability distributions. At any moment of time, such robot a is at a probability distribution over the locations of the grid. In the quantum case, one has a "quantum robot" [4] which can be in a quantum superposition of locations (i, j) and is allowed to perform transformations that move it at most one step at a time.

There are several ways to make this model of a "quantum robot" precise [1] and they all lead to similar results.

The simplest to define is the Z-local model of [1]. In this model, the robot's state space is spanned by states $|i, j, a\rangle$ with i, j representing the current location and a being the internal memory of the robot. The robot's state $|\psi\rangle$ can be any quantum superposition of those: $|\psi\rangle = \sum_{i,j,a} \alpha_{i,j,a}|i, j, a\rangle$, where $\alpha_{i,j,a}$ are complex numbers such that $\sum_{i,j,a} |\alpha_{i,j,a}|^2 = 1$. In one step, the robot can either perform a query of the value at the current location or a Z-local transformation.

A query is a transformation that leaves i, j parts of a state $|i, j, a\rangle$ unchanged and modifies the a part in a way that depends only on the value $a_{i,j}$. A Z-local transformation is a transformation that maps any state $|i, j, a\rangle$ to a superposition that involves only states with robot being either at the same location or at one of 4 adjacent locations ($|i, j, b\rangle$, $|i-1, j, b\rangle$, $|i+1, j, b\rangle$, $|i, j-1, b\rangle$ or $|i, j+1, b\rangle$ where the content of the robot's memory b is arbitrary).

The problem generalizes naturally to d-dimensional grid of size $N^{1/d} \times N^{1/d} \times \cdots \times N^{1/d}$, with robot being allowed to query or move one step in one of d directions in one unit of time.

Key Results

This problem was first studied by Benioff [4] who considered the use of the usual quantum search algorithm, by Grover [8] in this setting. Grover's algorithm allows to search a collection of N items $a_{i,j}$ with $O(\sqrt{N})$ queries. However, it does not respect the structure of a grid. Between any two queries it performs a transformation that may require the robot to move from any location (i,j) to any other location (i', j'). In the robot model, where the robot in only allowed to move one step in one time unit, such transformation requires $O(\sqrt{N})$ steps to perform. Implementing Grover's algorithm, which requires $O(\sqrt{N})$ such transformations, therefore, takes $O(\sqrt{N}) \times O(\sqrt{N}) = O(N)$ time, providing no advantage over the naive classical algorithm.

The first algorithm improving over the naive use of Grover's search was proposed by Aaronson and Ambainis [1] who achieved the following results:

- Search on $\sqrt{N} \times \sqrt{N}$ grid, if it is known that the grid contains exactly one $a_{i,j} = 1$ in $O(\sqrt{N} \log^{3/2} N)$ steps.
- Search on $\sqrt{N} \times \sqrt{N}$ grid, if the grid may contain an arbitrary number of $a_{i,j} = 1$ in $O(\sqrt{N} \log^{5/2} N)$ steps.
- Search on $N^{1/d} \times N^{1/d} \times \cdots \times N^{1/d}$ grid, for $d \geq 3$, in $O(\sqrt{N})$ steps.

They also considered a generalization of the problem, search on a graph G, in which the robot moves on the vertices v of the graph G and searches for a variable $a_v = 1$. In one step, the robot can examine the variable a_v corresponding to the current vertex v or move to another vertex w adjacent to v. Aaronson and Ambainis [1] gave an algorithm for searching an arbitrary graph with grid-like expansion properties in $O(N^{1/2+o(1)})$ steps. The main technique in those algorithms was the use of Grover's search and its generalization, amplitude amplification [5], in combination with "divide-and-conquer" methods recursively breaking up a grid into smaller parts.

The next algorithms were based on quantum walks [3,6,7]. Ambainis, Kempe and Rivosh [3] presented an algorithm, based on a discrete time quantum walk, which searches the two-dimensional $\sqrt{N} \times \sqrt{N}$ in $O(\sqrt{N} \log N)$ steps, if the grid is known to contain exactly one $a_{i,j} = 1$ and in $O(\sqrt{N} \log^2 N)$ steps in the general case. Childs and Goldstone [7] achieved a similar performance, using continuous time quantum walk. Curiously, it turned out that the performance of the walk crucially depended on the particular choice of the quantum walk, both in the discrete and continuous time and some very natural choices of quantum walk (e. g. one in [6]) failed.

Besides providing an almost optimal quantum speedup, the quantum walk algorithms also have an additional advantage: their simplicity. The discrete quantum walk algorithm of [3] uses just two bits of quantum memory. It's basis states are $|i, j, d\rangle$, where (i,j) is a location on the grid and d is one of 4 directions: $\leftarrow$, $\rightarrow$, $\uparrow$ and $\downarrow$. The basic algorithm consists of the following simple steps:

1. Generate the state $\sum_{i,j,d} \frac{1}{2\sqrt{N}} |i, j, d\rangle$.
2. $O(\sqrt{N \log N})$ times repeat
   (a) Perform the transformation
$$C_0 = \begin{pmatrix} -\frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & -\frac{1}{2} \end{pmatrix}$$
   on the states $|i, j, \leftarrow\rangle$, $|i, j, \rightarrow\rangle$, $|i, j, \uparrow\rangle$, $|i, j, \downarrow\rangle$, if $a_{i,j} = 0$ and the transformation $C_1 = -I$ on the same four states if $a_{i,j} = 1$.
   (b) Move one step according to the direction register and reverse the direction:
$$\begin{aligned} |i, j, \rightarrow\rangle &\rightarrow |i + 1, j, \leftarrow\rangle , \\ |i, j, \leftarrow\rangle &\rightarrow |i - 1, j, \rightarrow\rangle , \\ |i, j, \uparrow\rangle &\rightarrow |i, j - 1, \downarrow\rangle , \\ |i, j, \downarrow\rangle &\rightarrow |i, j + 1, \uparrow\rangle . \end{aligned}$$

In case, if $a_{i,j} = 1$ for one location (i,j), a significant part of the algorithm's final state will consist of the four states $|i, j, d\rangle$ for the location (i,j) with $a_{i,j} = 1$. This can be used to detect the presence of such location. A quantum algorithm for search on a grid can be also derived by designing a classical algorithm that finds $a_{i,j} = 1$ by performing a random walk on the grid and then applying Szegedy's general translation of classical random walks to quantum random chains, with a quadratic speedup over the classical random walk algorithm [12]. The resulting algorithm is similar to the algorithm of [3] described above and has the same running time.

For an overview on related quantum algorithms using similar methods, see [2,9].

Applications

Quantum algorithms for spatial search are useful for designing quantum communication protocols for the set disjointness problem. In the set disjointness problem, one has two parties holding inputs $x \in \{0, 1\}^N$ and $y \in \{0, 1\}^N$ and they have to determine if there is $i \in \{1, \ldots, N\}$ for which $x_i = y_i = 1$. (One can think of x and y as representing subsets $X, Y \subseteq \{1, \ldots, N\}$ with $x_i = 1 (y_i = 1)$ if $i \in X (i \in Y)$. Then, determining if $x_i = y_i = 1$ for some i is equivalent to determining if $X \cap Y \neq \emptyset$.)

The goal is to solve the problem, communicating as few bits between the two parties as possible. Classically, $\Omega(N)$ bits of communication are required [10]. The optimal quantum protocol [1] uses $O(\sqrt{N})$ quantum bits of communication and its main idea is to reduce the problem to spatial search. As shown by the $\Omega(\sqrt{N})$ lower bound of [11], this algorithm is optimal.

Cross References

▶ Quantization of Markov Chains
▶ Quantum Search

Recommended Reading

1. Aaronson, S., Ambainis, A.: Quantum search of spatial regions. In: Proc. 44th Annual IEEE Symp. on Foundations of Computer Science (FOCS), 2003, pp. 200–209
2. Ambainis, A.: Quantum walks and their algorithmic applications. Int. J. Quantum Inf. **1**, 507–518 (2003)
3. Ambainis, A., Kempe, J., Rivosh, A.: Coins make quantum walks faster. In: Proc. of SODA'05, pp 1099–1108
4. Benioff, P.: Space searches with a quantum robot. In: Quantum computation and information (Washington, DC, 2000). Contemp. Math., vol. 305, pp. 1–12. Amer. Math. Soc. Providence, RI (2002)
5. Brassard, G., Høyer, P., Mosca, M., Tapp, A.: Quantum amplitude amplification and estimation. In: Quantum computation and information (Washington, DC, 2000). Contemp. Math., vol. 305, pp. 53–74. American Mathematical Society, Providence, RI (2002)
6. Childs, A.M., Goldstone, J.: Spatial search by quantum walk. Phys. Rev. A **70**, 022314 (2004)
7. Childs, A.M., Goldstone, J.: Spatial search and the Dirac equation. Phys. Rev. A. **70**, 042312 (2004)
8. Grover, L.: A fast quantum mechanical algorithm for database search. In: Proc. 28th STOC, Philadelphia, Pennsylvania, pp 212–219. ACM Press, New York, (1996)
9. Kempe, J.: Quantum random walks – an introductory overview. Contemp. Phys. **44**(4), 302–327 (2003)
10. Razborov, A.: On the Distributional Complexity of Disjointness. Theor. Comput. Sci. **106**(2), 385–390 (1992)
11. Razborov, A.A.: Quantum communication complexity of symmetric predicates. Izvestiya of the Russian Academy of Science, Mathematics, **67**, 145–159 (2002)
12. Szegedy, M.: Quantum speed-up of Markov Chain based algorithms. In: Proceedings of FOCS'04, pp. 32–41

Quantum Algorithm for Solving the Pell's Equation
2002; Hallgren

SEAN HALLGREN
Department of Computer Science and Engineering, The Pennsylvania State University, University Park, PA, USA

Problem Definition

Pell's equation is one of the oldest studied problem in number theory. For a positive square-free integer d, Pell's equation is $x^2 - dy^2 = 1$, and the problem is to compute integer solutions x, y of the equation [7,9]. The earliest algorithm for it uses the continued fraction expansion of $\sqrt{d}$ and dates back to 1000 a.d. by Indian mathematicians. Lagrange showed that there are an infinite number of solutions of Pell's equation. All solutions are of the form $x_n + y_n\sqrt{d} = (x_1 + y_1\sqrt{d})^n$, where the smallest solution, (x_1, y_1), is called the fundamental solution. The solution (x_1, y_1) may have exponentially many bits in general in terms of the input size, which is $\log d$, and so cannot be written down in polynomial time. To resolve this difficulty, the computational problem is recast as computing the integer closest to the regulator $R = \ln(x_1 + y_1\sqrt{d})$. In this representation solutions of Pell's equation are positive integer multiples of R.

Solving Pell's equation is a special case of computing the unit group of number field. For a positive non-square integer Δ congruent to 0 or 1 mod 4, $K = \mathbb{Q}(\sqrt{\Delta})$ is a real quadratic number field. Its subring $\mathcal{O} = \mathbb{Z}[\frac{\Delta+\sqrt{\Delta}}{2}] \subseteq Q(\sqrt{\Delta})$ is called the quadratic order of discriminant Δ. The unit group is the set of invertible elements of $\mathcal{O}$. Units have the form $\pm\varepsilon^k$, where $k \in \mathbb{Z}$, for some $\varepsilon > 1$ called the fundamental unit. The fundamental unit ε can have exponentially many bits, so an approximation of the regulator $R = \ln\varepsilon$ is computed. In this representation the unit group consists of integer multiples of R. Given the integer closest to R there are classical polynomial-time algorithms to compute R to any precision. There are also efficient algorithms to test if a given number is a good approximation to an integer multiple of a unit, or to compute the least significant digits of $\varepsilon = e^R$ [1,3].

Two related and potentially more difficult problems are the principal ideal problem and computing the class group of a number field. In the principal ideal problem, a number field and an ideal I of $\mathcal{O}$ are given, and the problem is to decide if the ideal is principal, i. e. whether there exists α such that $I = \alpha\mathcal{O}$. If it is principal, then one can ask for an approximation of $\ln\alpha$. There are efficient classical algorithms to verify that a number is close to $\ln\alpha$ [1,3]. The class group of a number field is the finite abelian group defined by taking the set of fractional ideals modulo the principal fractional ideals. The class number is the size of the class group. Computing the unit group, computing the class group, and solving the principal ideal problems are three of the main problems of computational algebraic number theory [3]. Assuming the GRH, they are in NP∩CoNP [8].

Key Results

The best known classical algorithms for the problems defined in the last section take subexponential time, but there are polynomial-time quantum algorithms for them [4,6].

Theorem 1 *Given a quadratic discriminant Δ, there is a classical algorithm that computes an integer multiple of the regulator to within one. Assuming the GRH, this algorithm computes the regulator to within one and runs in expected time* $\exp(\sqrt{(\log \Delta) \log\log \Delta})^{O(1)}$.

Theorem 2 *There is a polynomial-time quantum algorithm that, given a quadratic discriminant Δ, approximates the regulator to within δ of the associated order $\mathcal{O}$ in time polynomial in* $\log \Delta$ *and* $\log \delta$ *with probability exponentially close to one.*

Corollary 1 *There is a polynomial-time quantum algorithm that solves Pell's equation.*

The quantum algorithm for Pell's equation uses the existence of a periodic function on the reals which has period R and is one-to-one within each period [4,6]. There is a discrete version of this function that can be computed efficiently. This function does not have the same periodic property since it cannot be evaluated at arbitrary real numbers such as R, but it does approximate the situation well enough for the quantum algorithm. In particular, computing the approximate period of this function gives R to the closest integer, or in other words, computes a generator for the unit group.

Theorem 3 *There is a polynomial-time quantum algorithm that solves the principal ideal problem in real quadratic number fields.*

Corollary 2 *There is a polynomial-time quantum algorithm that can break the Buchmann–Williams key-exchange protocol in real quadratic number fields.*

Theorem 4 *The class group and class number of a real quadratic number field can be computed in quantum polynomial-time assuming the GRH.*

Applications

Computationally hard number theoretic problems are useful for public key cryptosystems. There are reductions from factoring to Pell's equation and Pell's equation to the principal ideal problem, but no reductions are known in the opposite direction. The principal ideal problem forms the basis of the Buchmann–Williams key-exchange protocol [2]. Identification schemes based on this problem have been proposed by Hamdy and Maurer [5]. The classical exponential-time algorithms help determine which parameters to choose for the cryptosystem. The best known algorithm for Pell's equation is exponentially slower than the best factoring algorithm. Systems based on these harder problems were proposed as alternatives in case factoring turns out to be polynomial-time solvable. The efficient quantum algorithms can break these cryptosystems.

Open Problems

It remains open whether these problems can be solved in arbitrary degree number fields. The solution for Pell's equation can be thought of in terms of the hidden subgroup problem. That is, there exists a periodic function on the reals which has period $R \in \mathbb{R}$ and is one-to-one within each period. However, this function cannot be evaluated efficiently since it has an uncountable domain, and instead an efficiently computable approximation must be used. To evaluate this discrete version of the function, a classical algorithm is used to compute reduced ideals near a given point in $\mathbb{R}$. This algorithm is only polynomial-time for constant degree number fields as it computes the shortest vector in a lattice. Such an algorithm can be used to set up a superposition over points approximating the points in the a coset of the unit lattice. After setting up the superposition, it must shown Fourier sampling, i. e. computing the Fourier transform and measuring, suffices to compute the lattice.

Cross References

► Quantum Algorithm for Factoring
► Quantum Algorithms for Class Group of a Number Field

Recommended Reading

1. Buchmann, J., Thiel, C., Williams, H.C.: Short representation of quadratic integers. In: Bosma, W., van der Poorten A.J. (eds.) Computational Algebra and Number Theory, Sydney 1992. Mathematics and its Applications, vol. 325, pp. 159–185. Kluwer Academic Publishers (1995)
2. Buchmann, J.A., Williams, H.C.: A key exchange system based on real quadratic fields (extended abstract). In: Brassard, G. (ed.) Advances in Cryptology–CRYPTO '89. Lecture Notes in Computer Science, vol. 435, pp. 335–343. Springer 1990, 20–24 Aug (1989)
3. Cohen, H.: A course in computational algebraic number theory, vol. 138 of Graduate Texts in Mathematics. Springer (1993)
4. Hallgren, S.: Polynomial-time quantum algorithms for Pell's equation and the principal ideal problem. J. ACM **54**(1), 1–19 (2007)
5. Hamdy, S., Maurer, M.: Feige-fiat-shamir identification based on real quadratic fields, Tech. Report TI-23/99. Technische Universität Darmstadt, Fachbereich Informatik, http://www.informatik.tu-darmstadt.de/TI/Veroeffentlichung/TR/ (1999)
6. Jozsa, R.: Notes on Hallgren's efficient quantum algorithm for solving Pell's equation, tech. report, quant-ph/0302134 (2003)
7. Lenstra Jr, H.W.: Solving the Pell equation. Not. Am. Math. Soc. **49**, 182–192 (2002)

8. Thiel, C.: On the complexity of some problems in algorithmic algebraic number theory, Ph. D. thesis. Universität des Saarlandes, Saarbrücken, Germany (1995)
9. Williams, H.C.: Solving the Pell equation. In: Proc. Millennial Conference on Number Theory, pp. 397–435 (2002)

Quantum Approximation of the Jones Polynomial

2005; Aharonov, Jones, Landau

Zeph Landau
Department of Mathematics, City College of CUNY, New York, NY, USA

Keywords and Synonyms

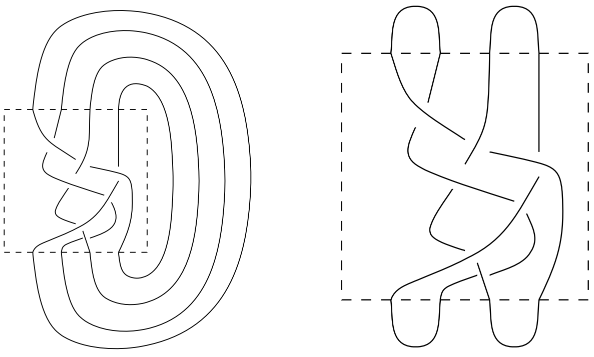
AJL algorithm

Problem Definition

A knot invariant is a function on knots (or links –i. e. circles embedded in R^3) which is invariant under isotopy of the knot, i. e., it does not change under stretching, moving, tangling, etc., (cutting the knot is not allowed). In low dimensional topology, the discovery and use of *knot invariants* is of central importance. In 1984, Jones [12] discovered a new knot invariant, now called the Jones polynomial $V_L(t)$, which is a Laurent polynomial in $\sqrt{t}$ with integer coefficients, and which is an invariant of the link L. In addition to the important role it has played in low dimensional topology, the Jones polynomial has found applications in numerous fields, from DNA recombination [16], to statistical physics [20].

From the moment of the discovery of the Jones polynomial, the question of how hard it is to compute became important. There is a very simple inductive algorithm (essentially due to Conway [5]) to compute it by changing crossings in a link diagram, but, naively applied, this takes exponential time in the number of crossings. It was shown [11] that the computation of $V_L(t)$ is #P-hard for all but a few values of t where $V_L(t)$ has an elementary interpretation. Thus a polynomial time algorithm for computing $V_L(t)$ for any value of t other than those elementary ones is unlikely. Of course, the #P-hardness of the problem does not rule out the possibility of good approximations. Still, the best classical algorithms to approximate the Jones polynomial at all but trivial values are exponential. Simply stated, the problem becomes:

Problem 1 *For what values of t and for what level of approximation can the Jones polynomial $V_L(t)$ be approximated in time polynomial in the number of crossings and links of the link L?*


Quantum Approximation of the Jones Polynomial, Figure 1
The trace closure (*left*) and plat closure (*right*) of the same 4-strand braid

Key Results

As mentioned above, exact computation of the Jones polynomial for most t is #P-hard and the best known classical algorithms to approximate the Jones polynomial are exponential. The key results described here consider the above problem in the context of quantum rather than classical computation.

The results concern the approximation of links that are given as closures of braids. (All links can be described this way.) Briefly, a braid of n strands and m crossings is described pictorially by n strands hanging alongside each other, with m crossings, each of two adjacent strands. A braid B may be "closed" to form a link by tying its ends together in a variety of ways, two of which are the *trace closure* (denoted by B^{tr}) which joins the ith strand from the top right to the ith strand from the bottom right (for each i), and the *plat closure* (denoted by B^{pl}) which is defined only for braids with an even number of strands by connecting pairs of adjacent strands (beginning at the rightmost strand) on both the top and bottom. Examples of the trace and plat closure of the same 4-strand braid are given in Fig. 1.

For such braids, the following results have been shown by Aharonov, Jones, and Landau:

Theorem 2.1 [3] *For a given braid B in B_n with m crossings, and a given integer k, there is a quantum algorithm which is polynomial in n,m,k which with all but exponentially (in n,m,k) small probability, outputs a complex number r with $|r - V_{B^{\mathrm{tr}}}(e^{2\pi i/k})| < \epsilon d^{n-1}$ where $d = 2\cos(\pi/k)$, and ϵ is inverse polynomial in n,k,m.*

Theorem 2.2 [3] *For a given braid B in B_n with m crossings, and a given integer k, there is a quantum algorithm which is polynomial in n,m,k which with all but exponentially (in n,m,k) small probability, outputs a complex* *number r with $|r - V_{B^{pl}}(e^{2\pi i/k})| < \epsilon d^{n/2-1}$ where $d = 2\cos(\pi/k)$ and ϵ is inverse polynomial in n,k,m.*

The original connection between quantum computation and the Jones polynomial was made earlier in the series of papers [6,7,8,9]. A model of quantum computation based on Topological Quantum Field Theory (*TQFT*) and Chern–Simons theory was defined in [6,7], and Kitaev, Larsen, Freedman and Wang showed that this model is polynomially equivalent in computational power to the standard quantum computation model in [8,9]. These results, combined with a deep connection between *TQFT* and the value of the Jones polynomial at particular roots of unity discovered by Witten 13 years earlier [18], implicitly implied (without explicitly formulating) an efficient quantum algorithm for the approximation of the Jones polynomial at the value $e^{2\pi i/5}$.

The approximation given by the above algorithms are additive, namely the result lies in a given window, whose size is independent of the actual value being approximated. The formulation of this kind of additive approximation was given in [4]; this is much weaker than a multiplicative approximation, which is what one might desire (again, see discussion in [4]). One might wonder if under such weak requirements, the problem remains meaningful at all. It turns out that, in fact, this additive approximation problem is hard for quantum computation, a result originally shown by Freedman, Kitaev, and Wang:

Theorem 2.3 Adapted from [9] *The problem of approximating the Jones polynomial of the plat closure of a braid at $e^{2\pi i/k}$ for constant k, to within the accuracy given in Theorem 2.2, is BQP-hard.*

A different proof of this result was given in [19], and the result was strengthened by Aharonov and Arad [1] to any k which is polynomial in the size of the input, namely, for all the plat closure cases for which the algorithm is polynomial in the size of the braid.

Understanding the Algorithm

The structure of the solution described by Theorems 2.1 and 2.2 consists of four steps:

1. *Mapping the Jones polynomial computation to a computation in the Temperley–Lieb algebra.* There exists a homomorphism of the braid group inside the so called Temperley–Lieb algebra (this homomorphism was the connection that led to the original discovery of the Jones polynomial in [12]). Using this homomorphism, the computation of the Jones polynomial of either the plat or trace closure of a braid can be mapped to the computation of a particular linear functional (called the Markov trace) of the image of the braid in the Temperley–Lieb algebra (for an essential understanding of a geometrical picture of the Temperley–Lieb algebra, see [14]).
2. *Mapping the Temperley–Lieb algebra calculation into a linear algebra calculation.* Using a representation of the Temperley–Lieb algebra, called the path model representation, the computation in step 1 is shown to be equal to a particular weighted trace of the matrix corresponding to the Temperley–Lieb algebra element coming from the original braid.
3. *Choosing the parameter t corresponding to unitary matrices.* The matrix in step 2 is a product of basic matrices corresponding to individual crossings in the braid group; an important characteristic of these basic matrices is that they have a local structure. In addition, by choosing the values of t as in Theorems 2.1 and 2.2, the matrices corresponding to individual crossings become unitary. The result is that the original problem has been turned into a weighted trace calculation of a matrix formed from a product of local unitary matrices–a problem well suited to a quantum computer.
4. *Implementing the quantum algorithm.* Finally the weighted trace calculation of a matrix described in step 3 is formally encoded into a calculation involving local unitary matrices and qubits.

A nice exposition of the algorithm is given in [15].

Applications

Since the publication [3], a number of interesting results have ensued investigating the possibility of quantum algorithms for other combinatorial/topological questions. Quantum algorithms have been developed for the case of the HOMFLYPT two-variable polynomial of the trace closure of a braid at certain pairs of values [19]. (This paper also extends the results of [3] to a class of more generalized braid closures; it is recommended reading as a complement to [3] or [15] as it gives the representation theory of the Jones-Wentzl representations thus putting the path model representation of the Temperley–Lieb algebra in a more general context). A quantum algorithm for the colored Jones polynomial is given in [10].

Recently, significant progress was made on the question of approximating the partition function of the Tutte polynomial of a graph [2]. This polynomial, at various parameters, captures important combinatorial features of the graph. Intimately associated to the Tutte polynomial is the Potts model, a model originating in statistical physics as a generalization of the Ising model to more than 2 states [17,20]; approximating the partition function of the Tutte polynomial of a graph is a very important question in statistical physics. The work of [2] develops a quantum algorithm for additive approximation of the Tutte polynomial for all planar graphs at all points in the Tutte plane and shows that for a significant set of these points (though not those corresponding to the Potts model) the problem of approximating is a complete problem for a quantum computer. Unlike previous results, these results use non-unitary representations.

Open Problems

There remain many unanswered questions related to the computation of the Jones polynomial from both a classical and quantum computational point of view.

From a classical computation point of view, the originally stated Problem 1 remains wide open for all but trivial choices of t. A result as strong as Theorem 2.2 for a classical computer seems unlikely since it would imply (via Theorem 2.3) that classical computation is as strong as quantum computation. A recent result by Jordan and Shor [13] shows that the approximation given in Theorem 2.1 solves a complete problem for a presumed (but not proven) weaker quantum model called the one clean qubit model. Since this model seems weaker than the full quantum computation model, a classical result as strong as Theorem 2.1 for the trace closure of a braid is perhaps in the realm of possibility.

From a quantum computational point of view, various open directions seem worthy of pursuit. Most of the quantum algorithms known as of the writing of this entry are based on the quantum Fourier transform, and solve problems which are algebraic and number theoretical in nature. Arguably, the greatest challenge in the field of quantum computation, (together with the physical realization of large scale quantum computers), is the design of new quantum algorithms based on substantially different techniques. The quantum algorithm to approximate the Jones polynomial is significantly different from the known quantum algorithms in that it solves a problem which is combinatorial in nature, and it does so without using the Fourier transform. These observations suggest investigating the possibility of quantum algorithms for other combinatorial/topological questions. Indeed, the results described in the applications section above address questions of this type. Of particular interest would be progress beyond [2] in the direction of the Potts model; specifically either showing that the approximation given in [2] is non-trivial or providing a different non-trivial algorithm.

Cross References

► Fault-Tolerant Quantum Computation
► Quantum Error Correction

Recommended Reading

1. Aharonov, D., Arad, I.: The BQP-hardness of approximating the Jones Polynomial. arxiv: quant-ph/0605181 (2006)
2. Aharonov, D., Arad, I., Eban, E., Landau, Z.: Polynomial Quantum Algorithms for Additive approximations of the Potts model and other Points of the Tutte Plane. arxiv:quant-ph/0702008 (2007)
3. Aharonov, D., Jones, V., Landau, Z.: A polynomial quantum algorithm for approximating the Jones polynomial. Proceedings of the 38th ACM Symposium on Theory of Computing (STOC) Seattle, Washington, USA, arxiv:quant-ph/0511096 (2006)
4. Bordewich, M., Freedman, M., Lovasz, L., Welsh, D.: Approximate counting and Quantum computation, Combinatorics. Prob. Comput. **14**(5–6), 737–754 (2005)
5. Conway, J.H.: An enumeration of knots and links, and some of their algebraic properties. Computational Problems in Abstract Algebra (Proc. Conf., Oxford, 1967), 329–358 (1970)
6. Freedman, M.: P/NP and the quantum field computer. Proc. Natl. Acad. Sci. USA **95**, 98–101 (1998)
7. Freedman, M., Kitaev, A., Larsen, M., Wang, Z.: Topological quantum computation. Mathematical challenges of the 21st century. (Los Angeles, CA, 2000). Bull. Amer. Math. Soc. (N.S.) **40**(1), 31–38 (2003)
8. Freedman, M.H., Kitaev, A., Wang, Z.: Simulation of topological field theories by quantum computers. Commun. Math. Phys. **227**, 587–603 (2002)
9. Freedman, M.H., Kitaev, A., Wang, Z.: A modular Functor which is universal for quantum computation. Commun. Math. Phys. **227**(3), 605–622 (2002)
10. Garnerone, S., Marzuoli, A., Rasetti, M.: An efficient quantum algorithm for colored Jones polynomials arXiv.org:quant-ph/0606167 (2006)
11. Jaeger, F., Vertigan, D., Welsh, D.: On the computational complexity of the Jones and Tutte polynomials. Math. Proc. Cambridge Philos. Soc. **108**(1), 35–53 (1990)
12. Jones, V.F.R.: A polynomial invariant for knots via von Neumann algebras. Bull. Am. Math. Soc. **12**(1), 103–111 (1985)
13. Jordan, S., Shor, P.: Estimating Jones polynomials is a complete problem for one clean qubit. http://arxiv.org/abs/0707.2831 (2007)
14. Kauffman, L.: State models and the Jones polynomial. Topology **26**, 395–407 (1987)
15. Kauffman, L., Lomonaco, S.: Topological Quantum Computing and the Jones Polynomial, arXiv.org:quant-ph/0605004 (2006)
16. Podtelezhnikov, A., Cozzarelli, N., Vologodskii, A.: Equilibrium distributions of topological states in circular DNA: interplay of supercoiling and knotting. (English. English summary) Proc. Natl. Acad. Sci. USA **96**(23), 12974–129 (1999)
17. Potts, R.: Some generalized order - disorder transformations, Proc. Camb. Phil. Soc. **48**, 106–109 (1952)
18. Witten, E.: Quantum field theory and the Jones polynomial. Commun. Math. Phys. **121**(3), 351–399 (1989)
19. Wocjan, P., Yard, J.: The Jones polynomial: quantum algorithms and applications in quantum complexity theory. In: Quantum Information and Computation, vol. 8, no. 1 & 2, 147–180 (2008). arXiv.org:quant-ph/0603069 (2006)
20. Wu, F.Y.: Knot Theory and statistical mechanics. Rev. Mod. Phys. **64**(4), 1099–1131 (1992)

Quantum Dense Coding

1992; Bennett, Wiesner

Barbara M. Terhal
IBM Research, Yorktown Heights, NY, USA

Keywords and Synonyms

Super dense coding; Dense coding

Problem Definition

Quantum information theory distinguishes classical bits from quantum bits or qubits. The quantum state of n qubits is represented by a complex vector in $(\mathbb{C}^2)^{\otimes n}$, where $(\mathbb{C}^2)^{\otimes n}$ is the tensor product of n 2-dimensional complex vector spaces. Classical n-bit strings form a basis for the vector space $(\mathbb{C}^2)^{\otimes n}$. Column vectors in $(\mathbb{C}^2)^{\otimes n}$ are denoted as $|\psi\rangle$ and row vectors are denoted as $|\psi\rangle^\dagger = |\psi\rangle^{*T} \equiv \langle\psi|$. The complex inner-product between vectors $|\psi\rangle$ and $|\phi\rangle$ is conveniently written as $\langle\psi|\phi\rangle$.

Entangled quantum states $|\psi\rangle \in (\mathbb{C}^2)^{\otimes n}$ are those quantum states that cannot be written as a product of some vectors $|\psi_i\rangle \in \mathbb{C}^2$, that is $|\psi\rangle \neq \bigotimes_i |\psi_i\rangle$. The Bell states are four orthogonal (maximally) entangled states defined as

$$|\Psi_{00}\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) , \quad |\Psi_{10}\rangle = \frac{1}{\sqrt{2}}(|00\rangle - |11\rangle) ,$$
$$|\Psi_{01}\rangle = \frac{1}{\sqrt{2}}(|01\rangle + |10\rangle) , \quad |\Psi_{11}\rangle = \frac{1}{\sqrt{2}}(|01\rangle - |10\rangle) .$$

The Pauli-matrices X, Y and Z are three unitary, Hermitian 2×2 matrices. They are defined as $X = |0\rangle\langle 1| + |1\rangle\langle 0|$, $Z = |0\rangle\langle 0| - |1\rangle\langle 1|$ and $Y = iXZ$.

Quantum states can evolve dynamically under inner-product preserving unitary operations U ($U^{-1} = U^\dagger$). Quantum information can be mapped onto observable classical information through the formalism of quantum measurements. In a quantum measurement on a state $|\psi\rangle$ in $(\mathbb{C}^2)^{\otimes n}$ a basis $\{|x\rangle\}$ in $(\mathbb{C}^2)^{\otimes n}$ is chosen. This basis is made observable through an interaction of the qubits with a macroscopic measurement system. A basis vector x is thus observed with probability $\mathbb{P}(x) = |\langle x|\psi\rangle|^2$.

Quantum information theory or more narrowly quantum Shannon theory is concerned with protocols which enable distant parties to efficiently transmit quantum or classical information, possibly aided by the sharing of quantum entanglement between the parties. For a detailed introduction to quantum information theory, see the book by Nielsen & Chuang [10].

Key Results

Super Dense Coding [3] is the protocol in which two classical bits of information are sent from sender Alice to receiver Bob. This is accomplished by sharing a Bell state $|\Psi_{00}\rangle_{AB}$ between Alice and Bob and the transmission of one qubit. The protocol is illustrated in Fig. 1. Given two bits b_1, b_2 Alice performs the following unitary transformation on her half of the Bell state:

$$P_{b_1 b_2} \otimes I_B |\Psi_{00}\rangle = |\Psi_{b_1 b_2}\rangle , \tag{1}$$

i. e. one of the four Bell states. Here $P_{00} = I, P_{01} = X$, $P_{10} = Z$ and $P_{11} = XZ = -iY$. Alice then sends her qubit to Bob. This allows Bob to do a measurement in the Bell basis. He distinguishes the four states $|\Psi_{b_1 b_2}\rangle$ and learns the value of the two bits b_1, b_2.

The protocol demonstrates the interplay between classical information and quantum information. No information can be communicated by merely sharing an entangled state such as $|\Psi_{00}\rangle$ without the actual transmission of physical information carriers. On the other hand it is a consequence of Holevo's theorem [8] that one qubit can encode at most one classical bit of information. The protocol of dense coding shows that the two resources of entanglement and qubit transmission *combined* give rise to a *super-dense coding* of classical information. Dense Coding is thus captured by the following resource inequality

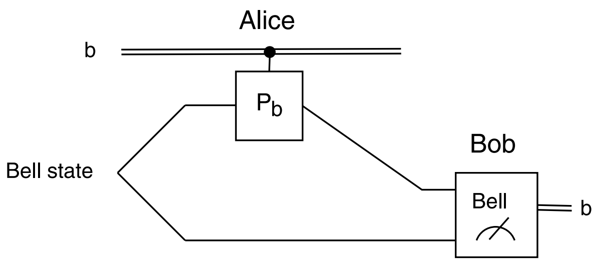
$$1 \text{ ebit} + 1 \text{ qubit} \succeq 2 \text{ cbits} . \tag{2}$$

In words, one bit of quantum entanglement (one ebit) in combination with the transmission of one qubit is sufficient for the transmission of two classical bits or cbits.

Quantum Teleportation [1] is a protocol that is dual to Dense Coding. In quantum teleportation, 1 ebit (a Bell state) is used in conjunction with the transmission of two classical bits to send one qubit from Alice to Bob. Thus the resource relation for Quantum Teleportation is

$$1 \text{ ebit} + 2 \text{ cbits} \succeq 1 \text{ qubit} . \tag{3}$$

The relation with quantum teleportation allows one to argue that dense coding is optimal. It is not possible to encode $2k$ classical bits in less than $m < k$ quantum bits even in the presence of shared quantum entanglement. Let us assume the opposite and obtain a contradiction. One uses quantum teleportation to convert the transmission of k quantum bits into the transmission of $2k$ classical bits. Then one can use the assumed super-dense coding scheme to encode these $2k$ bits into $m < k$ qubits. As a result one can send k quantum bits by effectively transmitting $m < k$ quantum bits (and sharing quantum entanglement) which is known to be impossible.



Quantum Dense Coding, Figure 1

Dense Coding. Alice and Bob use a shared Bell state to transmit two classical bits $b = (b_1, b_2)$ by sending one qubit. *Double lines* are classical bits and *single lines* represent quantum bits

Applications

Harrow [7] has introduced the notion of a coherent bit, or cobit. The notion of a cobit is useful in understanding resource relations and trade-offs between quantum and classical information. The noiseless transmission of a qubit from Alice to Bob can be viewed as the linear map $S_q\colon |x\rangle_A \rightarrow |x\rangle_B$ for a set of basis states $\{|x\rangle\}$. The transmission of a classical bit can be viewed as the linear map $S_c\colon |x\rangle_A \rightarrow |x\rangle_B|x\rangle_E$ where E stands for the environment Eve. Eve's copy of every basis state $|x\rangle$ can be viewed as the output of a quantum measurement and thus Bob's state is classical. The transmission of a cobit corresponds to the linear map $S_{co}\colon |x\rangle_A \rightarrow |x\rangle_A|x\rangle_B$. Since Alice keeps a copy of the transmitted data, Bob's state is classical. On the other hand, the cobit can also be used to generate a Bell state between Alice and Bob. Since no qubit can be transmitted via a cobit, a cobit is weaker than a qubit. A cobit is stronger than a classical bit since entanglement can be generated using a cobit.

One can define a *coherent* version of super-dense coding and quantum teleportation in which measurements are replaced by unitary operations. In this version of dense coding Bob replaces his Bell measurement by a rotation of the states $|\Psi_{b_1b_2}\rangle$ to the states $|b_1b_2\rangle_B$. Since Alice keeps her input bits, the coherent protocol implements the map $|x_1x_2\rangle_A \rightarrow |x_1x_2\rangle_A|x_1x_2\rangle_B$. Thus one can strengthen the dense coding resource relation to

$$1\text{ ebit} + 1\text{ qubit} \succeq 2\text{ cobits}\,. \qquad (4)$$

Similarly, the coherent execution of quantum teleportation gives rise to the modified relation 2 cobits + 1 ebit $\succeq$ 1 qubit + 2 ebits. One can omit 1 ebit on both sides of the inequality by using ebits catalytically, i. e. they can be borrowed and returned at the end of the protocol. One can then combine both coherent resource inequalities and obtain a resource *equality*

$$2\text{ cobits} = 1\text{ qubit} + 1\text{ ebit}\,. \qquad (5)$$

A different extension of dense coding is the notion of super-dense coding of quantum states proposed in [6]. Instead of dense coding classical bits, the authors in [6] propose to code quantum bits *whose quantum states are known to the sender Alice*. This last restriction is usually referred to as the remote preparation of qubits, in contrast to the transmission of qubits whose states are unknown to the sender. In remote preparation of qubits the sender Alice can use the additional knowledge about her states in the choice of encoding. In [6] it is shown that one can obtain the asymptotic resource relation

$$1\text{ ebit} + 1\text{ qubit} \succeq 2\text{ remotely prepared qubit(s)}\,. \qquad (6)$$

Such relation would be impossible if the r.h.s. were replaced by 2 qubits. In that case the inequality could be used repeatedly to obtain that 1 qubit suffices for the transmission of an arbitrary number of qubits which is impossible.

The "non-oblivious" super-dense coding of quantum states should be compared with the non-oblivious and asymptotic variant of quantum teleportation which was introduced in [2]. In this protocol, referred to as remote state preparation (using classical bits), the quantum teleportation inequality, Eq. (3) is tightened to

$$1\text{ ebit} + 1\text{ cbit} \succeq 1\text{ remotely prepared qubit(s)}\,. \qquad (7)$$

These various resource (in)equalities and their underlying protocols can be viewed as the first in a comprehensive theory of resources inequalities. The goal of such theory [4] is to provide a unified and simplified approach to quantum Shannon theory.

Experimental Results

In [9] a partial realization of dense coding was given using polarization states of photons as qubits. The Bell state $|\Psi_{01}\rangle$ can be produced by parametric down-conversion; this state was used in the experiment as the shared entanglement between Alice and Bob. With current experimental techniques it is not possible to carry out a low-noise measurement in the Bell basis which uniquely distinguishes the four Bell states. Thus in [9] one of three messages, *a trit*, is encoded into the four Bell states. Using two-particle interferometry Bob learns the value of the trit by distinguishing two of the four Bell states uniquely and obtaining a third measurement signal for the two other Bell states.

In perfect dense coding the channel capacity is 2 bits. For the trit-scheme of [9] the ideal channel capacity is $\log 3 \approx 1.58$. Due to the noise in the operations and measurements the authors of [9] estimate the experimentally achieved capacity as 1.13 bits.

In [11] the complete protocol of dense coding was carried out using two $^9\mathrm{Be}^+$ ions confined to an electromagnetic trap. A qubit is formed by two internal hyperfine levels of the $^9\mathrm{Be}^+$ ion. Single qubit and two-qubit operations are carried out using two polarized laser beams. A single qubit measurement is performed by observing a weak/strong fluorescence of $|0\rangle$ and $|1\rangle$. The authors estimate that the noise in the unitary transformations and measurements leads to an overall error rate on the transmission of the bits b of 15%. This results in an effective channel capacity of 1.16 bits.

In [5] dense coding was carried out using NMR spectroscopy. The two qubits were formed by the nuclear spins of ^{1}H and ^{13}C of chloroform molecules $^{13}\mathrm{CHCL}_3$ in liquid solution at room temperature. The full dense coding protocol was implemented using the technique of temporal averaging and the application of coherent RF pulses, see [10] for details. The authors estimate an overall error-rate on the transmission of the bits b of less than 10%.

Cross References

▶ Teleportation of Quantum States

Recommended Reading

1. Bennett, C.H., Brassard, G., Crepeau, C., Jozsa, R., Peres, A., Wootters, W.K.: Teleporting an unknown quantum state via dual classical and Einstein-Podolsky-Rosen channels. Phys. Rev. Lett. **70**, 1895–1899 (1993)
2. Bennett, C.H., DiVincenzo, D.P., Smolin, J.A., Terhal, B.M., Wootters, W.K.: Remote state preparation. Phys. Rev. Lett. **87**, 077902 (2001)
3. Bennett, C.H., Wiesner, S.J.: Communication via one- and two-particle operators on Einstein-Podolsky-Rosen states. Phys. Rev. Lett. **69**, 2881–2884 (1992)
4. Devetak, I., Harrow, A., Winter, A.: A resource framework for quantum Shannon theory. Tech. Report CSTR-05-008, CS Department, University of Bristol, December (2005)
5. Fang, X., Zhu, X., Feng, M., Mao, X., Du, F.: Experimental implementation of dense coding using nuclear magnetic resonance. Phys. Rev. A **61**, 022307 (2000)
6. Harrow, A., Hayden, P., Leung, D.: Superdense coding of quantum states. Phys. Rev. Lett. **92**, 187901 (2004)
7. Harrow, A.W.: Coherent communication of classical messages. Phys. Rev. Lett. **92**, 097902 (2004)
8. Holevo, A.S.: Bounds for the quantity of information transmitted by a quantum communication channel. Problemy Peredachi Informatsii, **9**, 3–11 (1973). English translation in: Probl. Inf. Transm. **9**, 177–183 (1973)
9. Mattle, K., Weinfurter, H., Kwiat, P.G., Zeilinger, A.: Dense coding in experimental quantum communication. Phys. Rev. Lett. **76**, 4656–4659 (1996)
10. Nielsen, M.A., Chuang, I.L.: Quantum computation and quantum information. Cambridge University Press, Cambridge, U.K. (2000)
11. Schaetz T., Barrett, M.D., Leibfried, D., Chiaverini, J., Britton, J., Itano, W.M., Jost, J.D., Langer, C., Wineland, D.J.: Quantum Dense Coding with Atomic Qubits. Phys. Rev. Lett. **93**, 040505 (2004)

Quantum Error Correction

1995; Shor

Martin Rötteler
NEC Laboratories America, Princeton, NJ, USA

Keywords and Synonyms

Quantum error-correcting codes; Quantum codes; Stabilizer codes

Problem Definition

Quantum systems can never be considered isolated from an environment which permanently causes disturbances of the state of the system. This noise problem threatens quantum computers and their great promise, namely to provide a computational advantage over classical computers for certain problems (see also the cross references in the Sect.“Cross References”). Quantum noise is usually modeled by the notion of a *quantum channel* which generalizes the classical case, and, in particular, includes scenarios for communication (space) and storage (time) of quantum information. For more information about quantum channels and quantum information in general, see [12]. A basic channel is the quantum mechanical analog of the classical binary symmetric channel [11]. This quantum channel is called the *depolarizing channel* and depends on a parameter p. Its effect is to randomly apply one of the Pauli spin matrices X, Y, and Z to the state of the system, mapping a quantum state ρ of one qubit to $(1-p)\rho + p/3(X\rho X + Y\rho Y + Z\rho Z)$. It should be noted that it is always possible to map any quantum channel to a depolarizing channel by twirling operations. The basic problem of quantum error correction is to devise a mechanism which allows to perfectly recover quantum information which has been sent through a quantum channel, in particular the depolarizing channel.

Key Results

For a long time, it was not known whether it would be possible to protect quantum information against noise. Even some indication in the form of the no-cloning theorem was put forward to support the view that it might be impossible. The no-cloning theorem essentially says that an unknown quantum state cannot be copied perfectly, thereby dashing the hopes that a simple triple-replication and majority voting mechanism (which works well classically) could be used for the quantum case. Therefore it came as a surprise when Shor [13] found a quantum code which encodes one qubit into nine qubits in such a way that the resulting state has the ability to be protected against arbitrary single-qubit errors on each of these nine qubits. The idea is to use a concatenation of two threefold repetition codes. One of them protects against bit-flip errors while the other protects against phase-flip errors. The quantum code is a two-dimensional subspace of the 2^9 dimensional Hibert space $(\mathbb{C}^2)^{\otimes 9}$. Two orthogonal basis vectors of this space are identified with the logical 0 and 1 states, respectively, usually called $|\underline{0}\rangle$ and $|\underline{1}\rangle$. Explicitly, the code is given by

$$
\begin{aligned}
|\underline{0}\rangle &= \frac{1}{2\sqrt{2}}(|000\rangle + |111\rangle) \otimes (|000\rangle + |111\rangle) \\
&\qquad \otimes (|000\rangle + |111\rangle)\,, \\
|\underline{1}\rangle &= \frac{1}{2\sqrt{2}}(|000\rangle - |111\rangle) \otimes (|000\rangle - |111\rangle) \\
&\qquad \otimes (|000\rangle - |111\rangle)\,.
\end{aligned}
$$

The state $\alpha|0\rangle + \beta|1\rangle$ of one qubit is encoded to the state $\alpha|\underline{0}\rangle + \beta|\underline{1}\rangle$ of the nine qubit system. The reason why this code can correct one arbitrary quantum error is as follows.

First, suppose that a bit-flip error has happened, which in quantum mechanical notation is given by the operator X. Then a majority vote of each block of three qubits $1-3, 4-6$, and $7-9$ can be computed and the bit-flip can be corrected. To correct against phase-flip errors, which are given by the operator Z, the fact is used that the code can be written as $|\underline{0}\rangle = |+++\rangle + |---\rangle$, $|\underline{1}\rangle = |+++\rangle - |---\rangle$, where $|\pm\rangle = \frac{1}{\sqrt{2}}(|000\rangle \pm |111\rangle)$. By measuring each block of three in the basis $\{|+\rangle, |-\rangle\}$, the majority of the phase-flips can be detected and one phase-flip error can be corrected. Similarly, it can be shown that Y, which is a combination of a bit-flip and a phase-flip, can be corrected.

Discretization of Noise

Even though the above procedure seemingly only takes care of bit-flips and phase-flip errors, it actually is true that an *arbitrary* error affecting a single qubit out of the nine qubits can be corrected. In particular, and perhaps surprisingly, this is also the case if one of the nine qubits is completely destroyed. The linearity of quantum mechanics allows this method to work. Linearity implies that whenever operators A and B can be corrected, so can their sum $A + B$ [6,13,15]. Since the (finite) set $\{\mathbf{1}_2, X, Y, Z\}$ forms a vector space basis for the (continuous) set of all one-qubit errors, the nine-qubit code can correct an arbitrary single qubit error.

Syndrome Decoding and the Need for Fresh Ancillas

A way to do the majority vote quantum-mechanically is to introduce two new qubits (also called ancillas) that are initialized in $|0\rangle$. Then, the results of the two parity checks for the repetition code of length three can be computed into these two ancillas. This syndrome computation for the repetition code can be done using the so-called controlled not (CNOT) gates [12] and Hadamard gates. After this, the qubits holding the syndrome will factor out (i. e., they have no influence on future superpositions or interferences of the computational qubits), and can be discarded. Quantum error correction demands a large supply of fresh qubits for the syndrome computations which have to be initialized in a state $|0\rangle$. The preparation of many such states is required to fuel active quantum error correcting cycles, in which syndrome measurements have to be applied repeatedly. This poses great challenges to any concrete physical realization of quantum error-correcting codes.

Conditions for General Quantum Codes

Soon after the discovery of the first quantum code, general conditions required for the existence of codes, which protect quantum systems against noise, were sought after. Here the noise is modeled by a general quantum channel, given by a set of error operators E_i. The Knill–Laflamme conditions [8] yield such a characterization. Let C be the code subspace and let P_C be an orthogonal projector onto C. Then the existence of a recovery operation for the channel with error operators E_i is equivalent to the equation

$$P_C E_i^\dagger E_j P_C = \lambda_{i,j} P_C\,,$$

for all i and j, where $\lambda_{i,j}$ are some complex constants. This recently has been extended to the more general framework of subsystem codes (also called operator quantum error correcting codes) [10].

Constructing Quantum Codes

The problem of deriving general constructions of quantum codes was addressed in a series of ground-breaking papers by several research groups in the mid 90s. Techniques were developed which allow classical coding theory to be imported to an extent that is enough to provide many families of quantum codes with excellent error correction properties.

The IBM group [2] investigated quantum channels, placed bounds on the quantum channels' capacities, and showed that for some channels it is possible to compute the capacity (such as for the quantum erasure channel). Furthermore, they showed the existence of a five qubit quantum code that can correct an arbitrary error, thereby being much more efficient than Shor's code. Around the same time, Calderbank and Shor [4] and Steane [14] found a construction of quantum codes from any pair C_1, C_2 of classical linear codes satisfying $C_2^{\perp} \subseteq C_1$. Named after their inventors, these codes are known as CSS codes.

The AT&T group [3] found a general way of defining a quantum code. Whenever a classical code over the finite field $\mathbb{F}_4$ exists that is additively closed and self-orthogonal with respect to the Hermitian inner product, they were able to find even more examples of codes. Independently, D. Gottesman [6,7] developed the theory of stabilizer codes. These are defined as the simultaneous eigenspaces of an abelian subgroup of the group of tensor products of Pauli matrices on several qubits. Soon after this, it was realized that the two constructions are equivalent.

A stabilizer code which encodes k qubits into n qubits and has distance d is denoted by $[[n, k, d]]$. It can correct up to $\lfloor (d-1)/2 \rfloor$ errors of the n qubits. The rate of the code is defined as $r = k/n$. Similar to classical codes, bounds on quantum error-correcting codes are known; i. e., the Hamming, Singleton, and linear programming bounds.

Asymptotically Good Codes

Matching the developments in classical algebraic coding theory, an interesting question deals with the existence of asymptotically good codes; i. e., families of quantum codes with parameters $[[n_i, k_i, d_i]]$, where $i \geq 0$, which have asymptotically non-vanishing rate $\lim_{i\to\infty} k_i/n_i > 0$ and non-vanishing relative distance $\lim_{i\to\infty} d_i/n_i > 0$. In [4], the existence of asymptotically good codes was established using random codes. Using algebraic geometry (Goppa) codes, it was later shown by Ashikhmin, Litsyn, and Tsfasman that there are also explicit families of asymptotically good quantum codes. Currently, most constructions of quantum codes are from the above mentioned stabilizer/additive code construction, with notable exception of a few non-additive codes and some codes which do not fit into the framework of Pauli error bases.

Applications

Besides their canonical application to protect quantum information against noise, quantum error correcting codes have been used for other purposes as well. The Preskill/Shor proof of the security of the quantum key distribution scheme BB84 relies on an entanglement purification protocol, which in turn uses CSS codes. Furthermore, quantum codes have been used for quantum secret sharing, quantum message authentication, and secure multiparty quantum computations. Properties of stabilizer codes are also germane for the theory of fault-tolerant quantum computation.

Open Problems

The literature of quantum error correction is fast growing, and the list of open problems is certainly too vast to be surveyed here in detail. The following short list is highly influenced by the preference of the author.

It is desirable to find quantum codes for which all stabilizer generators have low weight. This would be the quantum equivalent of low-density parity check (LDPC) codes. Since the weights directly translate into the complexity of the syndrome computation circuitry, it would be highly desirable to find examples of such codes. So far, only few sporadic constructions are known.

It is an open problem to find new families of quantum codes which improve on the currently known estimates on the threshold for fault-tolerant quantum computing. An advantage might be to use subsystem codes, since they allow for simple error correction circuits. It would be useful to find more families of subsystem codes, thereby generalizing the Bacon/Shor construction.

Most quantum codes are designed for the depolarizing channel, where – roughly speaking – the error probability is improved from p to $p^{d/2}$ for a distance d code. The independence assumption underlying this model might not always be justified and therefore it seems imperative to consider other, e. g., non-Markovian, error models. Under some assumptions on the decay of the interaction strengths, threshold results for such channels have been shown. However, good constructions of codes for such types of noise are still out of reach.

Approximate quantum error-correcting codes have found applications in quantum authentication and recently for secure multiparty quantum computations [1].

Here the Knill–Laflamme conditions do not have to be satisfied exactly, but some error is allowed. This gives much more freedom in defining subspaces and if some error can be tolerated, quantum codes with much better error correction capabilities become feasible. However, not many constructions of such codes are known.

Experimental Results

Active quantum error-correcting codes, such as those codes which require syndrome measurements and correction operations, as well as passive codes (i. e., codes in which the system stays in an simultaneous invariant subspace of all error operators for certain types of noise), have been demonstrated for some physical systems. The most advanced physical demonstration in this respect are the nuclear magnetic resonance (NMR) experiments [9]. The three-qubit repetition code, which protects one qubit against phase-flip error Z, was demonstrated in an ion-trap for beryllium ion qubits [5].

Data Sets

M. Grassl maintains http://www.codetables.de, which contains tables of the best known quantum codes, some entries of which extend [3, Table III]. It also contains bounds on the minimum distance of quantum codes for given lengths and dimensions, and contains information about the construction of the codes. In principle, this can be used to get explicit generator matrices (see also the following section, "URL to Code").

URL to Code

The computer algebra system Magma (http://magma.maths.usyd.edu.au/magma/) has functions and data structures for defining and analyzing quantum codes. Several quantum codes are already defined in a database of quantum codes. For instance, the command `QECC(F,n,k)` returns the best known quantum code (i. e., the one of highest distance) over the field F, of length n, and dimension k. It allows the user to define new quantum codes, to study their properties (such as the weight distribution, automorphism), and several predefined methods for obtaining new codes from old ones.

Cross References

- ► Abelian Hidden Subgroup Problem
- ► Fault-Tolerant Quantum Computation
- ► Quantization of Markov Chains
- ► Quantum Algorithm for Element Distinctness
- ► Quantum Algorithm for Factoring
- ► Quantum Algorithm for Finding Triangles
- ► Quantum Algorithm for Solving the Pell's Equation
- ► Quantum Key Distribution
- ► Teleportation of Quantum States

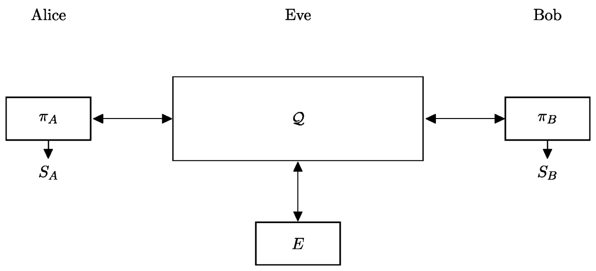
Recommended Reading

1. Ben-Or, M., Crépeau, C., Gottesman, D., Hassidim, A., Smith, A.: Secure multiparty quantum computation with (only) a strict honest majority. In: Proceedings of the 47th Symposium on Foundations of Computer Science (FOCS'06), 2006, pp. 249–260
2. Bennett, C.H., DiVincenzo, D.P., Smolin, J.A., Wootters, W.K.: Mixed-state entanglement and quantum error correction. Phys. Rev. A **54**, 3824–3851 (1996)
3. Calderbank, A.R., Rains, E.M., Shor, P.W., Sloane, N.J.A.: Quantum error correction via codes over GF(4). IEEE Trans. Inform. Theory **44**, 1369–1387 (1998)
4. Calderbank, A.R., Shor, P.W.: Good quantum error-correcting codes exist. Phys. Rev. A **54**, 1098–1105 (1996)
5. Chiaverini, J., Leibfried, D., Schaetz, T., Barrett, M.D., Blakestad, R.B., Britton, J., Itano, W.M., Jost, J.D., Knill, E., Langer, C., Ozeri, R., Wineland, D.J.: Realization of quantum error correction. Nature **432**, 602–605 (2004)
6. Gottesman, D.: Class of quantum error-correcting codes saturating the quantum Hamming bound. Phys. Rev. A **54**, 1862–1868 (1996)
7. Gottesman, D.: Stabilizer codes and quantum error correction, Ph. D. thesis, Caltech. (1997) See also: arXiv preprint quant-ph/9705052
8. Knill, E., Laflamme, R.: Theory of quantum error-correcting codes. Phys. Rev. A **55**, 900–911 (1997)
9. Knill, E., Laflamme, R., Martinez, R., Negrevergne, C.: Benchmarking quantum computers: the five-qubit error correcting code. Phys. Rev. Lett. **86**, 5811–5814 (2001)
10. Kribs, D., Laflamme, R., Poulin, D.: Unified and generalized approach to quantum error correction. Phys. Rev. Lett. **94**(4), 180501 (2005)
11. MacWilliams, F.J., Sloane, N.J.A.: The Theory of Error–Correcting Codes. North–Holland, Amsterdam (1977)
12. Nielsen, M., Chuang, I.: Quantum Computation and Quantum Information. Cambridge University Press, Cambridge (2000)
13. Shor, P.W.: Scheme for reducing decoherence in quantum computer memory. Phys. Rev. A **52**, R2493–R2496 (1995)
14. Steane, A.: Error correcting codes in quantum theory. Phys. Rev. Lett. **77**, 793–797 (1996)
15. Steane, A.: Multiple-particle interference and quantum error correction. Proc. R. Soc. London A **452**, 2551–2577 (1996)

Quantum Key Distribution

1984; Bennett, Brassard
1991; Ekert

RENATO RENNER
ETH, Institute for Theoretical Physics, Zurich, Switzerland



Quantum Key Distribution, Figure 1
A QKD protocol π consists of algorithms π_A and π_B for Alice and Bob, respectively. The algorithms communicate over a quantum channel $\mathcal{Q}$ that might be coupled to a system E controlled by an adversary. The goal is to generate identical keys S_A and S_B which are independent of E

Keywords and Synonyms

Quantum key exchange, Quantum key growing

Problem Definition

Secret keys, i. e., random bitstrings not known to an adversary, are a vital resource in cryptography (they can be used, e. g., for message encryption or authentication). The *distribution* of secret keys among distant parties, possibly only connected by insecure communication channels, is thus a fundamental cryptographic problem. *Quantum key distribution (QKD)* is a method to solve this problem using quantum communication. It relies on the fact that any attempt of an adversary to wiretap the communication would, by the laws of quantum mechanics, inevitably introduce disturbances which can be detected.

For the technical definition, consider a setting consisting of two honest parties, called *Alice* and *Bob*, as well as an adversary, *Eve*. Alice and Bob are connected by a quantum channel $\mathcal{Q}$ which might be coupled to a (quantum) system E controlled by Eve (see Fig. 1). In addition, it is assumed that Alice and Bob have some means to exchange classical messages *authentically*, that is, they can make sure that Eve is unable to (undetectably) alter classical messages during transmission. If only insecure communication channels are available, Alice and Bob can achieve this using an *authentication scheme* [15] which, however, requires a short initial key. This is why QKD is sometimes called *Quantum Key Growing*.

A *QKD protocol* $\pi = (\pi_A, \pi_B)$ is a pair of algorithms for Alice and Bob, producing classical outputs S_A and S_B, respectively. S_A and S_B take values in $\mathcal{S} \cup \{\perp\}$ where $\mathcal{S}$ is called *key space* and $\perp$ is a symbol (not contained in $\mathcal{S}$) indicating that no key can be generated. A QKD protocol π with key space $\mathcal{S}$ is said to be *perfectly secure on a channel* $\mathcal{Q}$ if, after its execution using communication over $\mathcal{Q}$, the following holds:

- $S_A = S_B$;
- if $S_A \neq \perp$ then S_A and S_B are uniformly distributed on $\mathcal{S}$ and independent of the state of E.

More generally, π is said to be *ε-secure on $\mathcal{Q}$* if it satisfies the above conditions except with probability (at most) ε. Furthermore, π is said to be *ε-robust on $\mathcal{Q}$* if the probability that $S_A = \perp$ is at most ε.

In the standard literature on QKD, protocols are typically parametrized by some positive number k quantifying certain resources needed for its execution (e. g., the amount of communication). A protocol $\pi = (\pi_k)_{k \in \mathbb{N}}$ is said to be *secure* (*robust*) on a channel $\mathcal{Q}$ if there exists a sequence $(\varepsilon_k)_{k \in \mathbb{N}}$ which approaches zero exponentially fast such that π_k is ε_k-secure (ε_k-robust) on $\mathcal{Q}$ for any $k \in \mathbb{N}$. Moreover, if the key space of π_k is denoted by $\mathcal{S}_k$, the *key rate* of $\pi = (\pi_k)_{k \in \mathbb{N}}$ is defined by $r = \lim_{k \to \infty} \frac{\ell_k}{k}$ where $\ell_k := \log_2 |\mathcal{S}_k|$ is the key length.

The ultimate goal is to construct QKD protocols π which are secure against general attacks, i. e., on *all* possible channels $\mathcal{Q}$. This ensures that an adversary cannot get any information on the generated key even if she fully controls the communication between Alice and Bob. At the same time, a protocol π should be robust on certain realistic (possibly noisy) channels $\mathcal{Q}$ in the absence of an adversary. That is, the protocol must always produce a key, unless the disturbances in the channel exceed a certain threshold. Note that, in contrast to security, robustness cannot be guaranteed in general (i. e., on all $\mathcal{Q}$), as an adversary could, for instance, interrupt the entire communication between Alice and Bob (in which case key generation is obviously impossible).

Key Results

Protocols

On the basis of the pioneering work of Wiesner [16], Bennett and Brassard, in 1984, invented QKD and proposed a first protocol, known today as the *BB84 protocol* [2]. The idea was then further developed by Ekert, who established a connection to quantum entanglement [7]. Later, in an attempt to increase the efficiency and practicability of QKD, various extensions to the BB84 protocol as well as alternative types of protocols have been proposed.

QKD protocols can generally be subdivided into (at least) two subprotocols. The purpose of the first, called *distribution protocol*, is to generate a *raw key pair*, i. e., a pair of correlated classical values X and Y known to Alice and Bob, respectively. In most protocols (including BB84), Alice chooses $X = (X_1, \ldots, X_k)$ at random, encodes each of the X_i into the state of a quantum particle, and then sends the k particles over the quantum channel to Bob. Upon receiving the particles, Bob applies a measurement to each of them, resulting in $Y = (Y_1, \ldots, Y_k)$. The crucial idea now is that, by virtue of the laws of quantum mechanics, the secrecy of the raw key is a function of the strength of the correlation between X and Y; in other words, the more information (on the raw) key an adversary tries to acquire, the more disturbances she introduces.

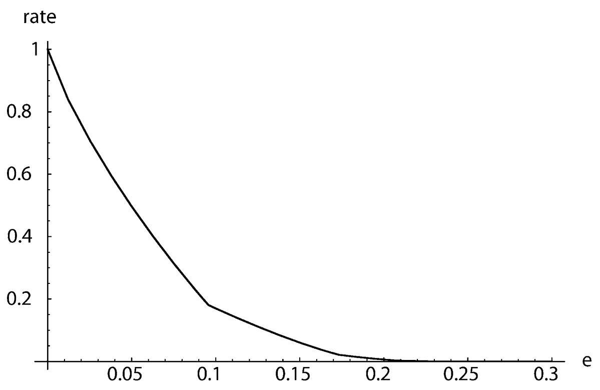
This is exploited in the second subprotocol, called *distillation protocol*. Roughly speaking, Alice and Bob estimate the statistics of the raw key pair (X, Y). If the correlation between their respective parts is sufficiently strong, they use classical techniques such as *information reconciliation* (error correction) and *privacy amplification* (see [3] for the case of a classical adversary which is relevant for the analysis of security against individual attacks and [12,13] for the quantum-mechanical case which is relevant in the context of collective and general attacks) to turn (X, Y) into a pair (S_A, S_B) of identical and secure keys.

Key Rate as a Function of Robustness and Security

The performance (in terms of the key rate) of a QKD protocol strongly depends on the desired level of security and robustness it is supposed to provide, as illustrated in Fig. 2. (The robustness is typically measured in terms of the *maximum tolerated channel noise*, i. e., the maximum noise of a channel $\mathcal{Q}$ such that the protocol is still robust on $\mathcal{Q}$ according to the above definition.) The results summarized below apply to protocols of the form described above where, for the analysis of robustness, it is assumed that the quantum channel $\mathcal{Q}$ connecting Alice and Bob is *memoryless* and *time-invariant*, i. e., each transmission is subject to the same type of disturbances. Formally, such channels are denoted by $\mathcal{Q} = \bar{\mathcal{Q}}^{\otimes k}$ where $\bar{\mathcal{Q}}$ describes the action of the channel in a single transmission.

Security Against Individual Attacks A QKD protocol π is said to be *secure against individual attacks* if it is secure on any channel $\mathcal{Q}$ of the form $\bar{\mathcal{Q}}^{\otimes k}$ where the coupling to E is purely classical. Note that this notion of security is relatively weak. Essentially, it only captures attacks where the adversary applies identical and independent measurements to each of the particles sent over the channel.

The following general statement can be derived from a classical argument due to Csiszár and Körner [5]. Let τ be a distribution protocol as described above, i. e., τ generates a raw key pair (X, Y). Moreover, let S be a set of quantum channels $\bar{\mathcal{Q}}$ suitable for τ. Then there exists a QKD protocol π (parametrized by k), consisting of k executions



Quantum Key Distribution, Figure 2
Key rate of an extended version of the BB84 QKD protocol depending on the maximum tolerated channel noise (measured in terms of the bit-flip probability *e*) [12]

of the subprotocol τ followed by an appropriate distillation protocol such that the following holds: π is robust on $\mathcal{Q} = \bar{\mathcal{Q}}^{\otimes k}$ for any $\bar{\mathcal{Q}} \in S$, secure against individual attacks, and has key rate at least

$$r \geq \min_{\bar{\mathcal{Q}} \in S} H(X|Z) - H(X|Y) , \tag{1}$$

where the Shannon entropies on the r.h.s. are evaluated for the joint distribution $P^{\bar{\mathcal{Q}}}_{XYZ}$ of the raw key (X, Y) and the (classical) value Z held by Eve's system E after one execution of τ on the channel $\bar{\mathcal{Q}}$. Evaluating the right hand side for the BB84 protocol on a channel with bit-flip probability e shows that the rate is non-negative if $e \leq 14.6\%$ [8].

Security Against Collective Attacks A QKD protocol π is said to be *secure against collective attacks* if it is secure on any channel $\mathcal{Q}$ of the form $\bar{\mathcal{Q}}^{\otimes k}$ with arbitrary coupling to E. This notion of security is strictly stronger than security against individual attacks, but it still relies on the assumption that an adversary does not apply joint operations to the particles sent over the channel.

As shown by Devetak and Winter [6], the above statement for individual attacks extends to collective attacks when replacing inequality (1) by

$$r \geq \min_{\bar{\mathcal{Q}} \in S} S(X|E) - H(X|Y) \tag{2}$$

where $S(X|E)$ is the conditional von Neumann entropy evaluated for the classical value X and the quantum state of E after one execution of τ on $\bar{\mathcal{Q}}$. For the standard BB84 protocol, the rate is positive as long as the bit-flip probability e of the channel satisfies $e \leq 11.0\%$ [14] (see Fig. 2 for a graph of the performance of an extended version of the protocol).

Security Against General Attacks A QKD protocol π is said to be *secure against general attacks* if it is secure on any arbitrary channel $\mathcal{Q}$. This type of security is sometimes also called *full* or *unconditional security* as it does not rely on any assumptions on the type of attacks or the resources needed by an adversary.

The first QKD protocol to be proved secure against general attacks was the BB84 protocol. The original argument by Mayers [11] was followed by various alternative proofs. Most notably, based on a connection to the problem of entanglement purification [4] established by Lo and Chau [10], Shor and Preskill [14] presented a general argument which applies to various versions of the BB84 protocol.

More recently it has been shown that, for virtually *any* QKD protocol, security against collective attacks implies security against general attacks [12]. In particular, the above statement about the security of QKD protocols against collective attacks, including formula 2 for the key rate, extends to security against general attacks.

Applications

Because the notion of security described above is *composable* [13] (see [1,12] for a general discussion of composability of QKD) the key generated by a secure QKD protocol can in principle be used within any application that requires a secret key (such as one-time pad encryption). More precisely, let $\mathcal{A}$ be a scheme which, when using a *perfect* key S (i. e., a uniformly distributed bitstring which is independent of the adversary's knowledge), has some failure probability δ (according to some arbitrary failure criterion). Then, if the perfect key S is replaced by the key generated by an ε-secure QKD protocol, the failure probability of $\mathcal{A}$ is bounded by $\delta + \varepsilon$ [13].

Experimental Results

Most known QKD protocols (including BB84) only require relatively simple quantum operations on Alice and Bob's side (e. g., preparing a two-level quantum system in a given state or measuring the state of such a system). This makes it possible to realize them with today's technology. Experimental implementations of QKD protocols usually use photons as carriers of quantum information, because they can easily be transmitted (e. g., through optical fibers). A main limitation, however, is noise in the transmission, which, with increasing distance between Alice and Bob, reduces the performance of the protocol (see Fig. 2). We refer to [9] for an overview on quantum cryptography with a focus on experimental aspects.

Cross References

▶ Quantum Error Correction
▶ Teleportation of Quantum States

Recommended Reading

1. Ben-Or, M., Horodecki, M., Leung, D.W., Mayers, D., Oppenheim, J.: The universal composable security of quantum key distribution. In: Second Theory of Cryptography Conference TCC. Lecture Notes in Computer Science, vol. 3378, pp. 386–406. Springer, Berlin (2005). Also available at http://arxiv.org/abs/quant-ph/0409078
2. Bennett, C.H., Brassard, G.: Quantum cryptography: Public-key distribution and coin tossing. In: Proceedings of IEEE International Conference on Computers, Systems and Signal Processing, pp. 175–179. IEEE Computer Society Press, Los Alamitos (1984)
3. Bennett, C.H., Brassard, G., Crépeau, C., Maurer, U.: Generalized privacy amplification. IEEE Trans. Inf. Theory **41**(6), 1915–1923 (1995)
4. Bennett, C.H., Brassard, G., Popescu, S., Schumacher, B., Smolin, J., Wootters, W.: Purification of noisy entanglement and faithful teleportation via noisy channels. Phys. Rev. Lett. **76**, 722–726 (1996)
5. Csiszár, I., Körner, J.: Broadcast channels with confidential messages. IEEE Trans. Inf. Theory **24**, 339–348 (1978)
6. Devetak, I., Winter, A.: Distillation of secret key and entanglement from quantum states. Proc. R. Soc. Lond. A **461**, 207–235 (2005)
7. Ekert, A.K.: Quantum cryptography based on Bell's theorem. Phys. Rev. Lett. **67**, 661–663 (1991)
8. Fuchs, C.A., Gisin, N., Griffiths, R.B., Niu, C., Peres, A.: Optimal eavesdropping in quantum cryptography, I. Information bound and optimal strategy. Phys. Rev. A **56**, 1163–1172 (1997)
9. Gisin, N., Ribordy, G., Tittel, W., Zbinden, H.: Quantum cryptography. Rev. Mod. Phys. **74**, 145–195 (2002)
10. Lo, H.-K., Chau, H.F.: Unconditional security of quantum key distribution over arbitrarily long distances. Science **283**, 2050–2056 (1999)
11. Mayers, D.: Quantum key distribution and string oblivious transfer in noisy channels. In: Advances in Cryptology – CRYPTO '96. Lecture Notes in Computer Science, vol. 1109, pp. 343–357. Springer (1996)
12. Renner, R.: Security of Quantum Key Distribution. Ph. D. thesis, Swiss Federal Institute of Technology (ETH) Zurich, Also available at http://arxiv.org/abs/quant-ph/0512258 (2005)
13. Renner, R., König, R.: Universally composable privacy amplification against quantum adversaries. In: Second Theory of Cryptography Conference TCC. Lecture Notes in Computer Science, vol. 3378, pp. 407–425. Springer, Berlin (2005). Also available at http://arxiv.org/abs/quant-ph/0403133
14. Shor, P.W., Preskill, J.: Simple proof of security of the BB84 quantum key distribution protocol. Phys. Rev. Lett. **85**, 441 (2000)
15. Wegman, M.N., Carter, J.L.: New hash functions and their use in authentication and set equality. *J. Comput. Syst. Sci.* **22**, 265–279 (1981)
16. Wiesner, S.: Conjugate coding. Sigact News **15**(1), 78–88 (1983)

Quantum Search

1996; Grover

Lov K. Grover[1], Ben W. Reichardt[2]
[1] Bell Labs, Alcatel-Lucent, Murray Hill, NJ, USA
[2] Institute for Quantum Information, California Institute of Technology, Pasadena, CA, USA

Keywords and Synonyms

Quantum unsorted database search

Problem Definition

Informal Description

The search problem can be described informally as, given a set of N items, identify an item satisfying a given property. Assume that it is easy to query whether a specific item satisfies the property or not. Now, the set of N items is not sorted and so there appears to be no shortcut to the brute-force method of checking each item one by one until the desired item is found. However, that intuition is only correct for classical computers; quantum computers can be in multiple states simultaneously and can examine multiple items at the same time. There is no obvious lower bound to how fast search can be run by a quantum computer, but nor is there an obvious technique faster than brute-force search. It turns out, though, that there is an efficient quantum mechanical search algorithm that makes only $O(\sqrt{N})$ queries, and this is optimal.

This quantum algorithm works very different from searching with a classical computer [5]. The optimal classical strategy is to check the items one at a time in random order. After η items are picked, the probability that the search hasn't yet succeeded is $(1-1/N)(1-1/(N-1))\cdots(1-1/(N-\eta+1))$. For $\eta \ll N$, the success probability is therefore roughly $1-(1-1/N)^{\eta} \approx \eta/N$. Increasing the success probability to a constant requires the number of items picked, η, to be $\Omega(N)$.

In contrast, the quantum search algorithm through a series of quantum mechanical operations steadily increases the amplitude on the target item. In η steps it increases this amplitude to roughly $\eta/\sqrt{N}$, and hence the success probability (on measuring the state) to η^2/N. Boosting this to $\Omega(1)$ requires only $O(\sqrt{N})$ steps, approximately the square-root of the number of steps required by any classical algorithm.

The reason the quantum search algorithm has been of so much interest in a variety of fields is that it can be adapted to different settings, giving a new class of quantum algorithms extending well beyond search problems.

Formal Statement

Given oracle access to a bit string $x \in \{0,1\}^N$, find an index i such that $x_i = 1$, if such index exists. In particular, determine if $x = 0^N$ or not – i. e., evaluate the OR function $x_1 \vee x_2 \vee \cdots \vee x_N$. To understand this, think of the indices i as combinatorial objects of some sort, and x_i indicates whether i satisfies a certain property or not – with x_i efficiently computable given i. The problem is to find an object satisfying the property. This search problem is *unstructured* because the solution may be arbitrary. Ordered search of a *sorted* list, on the other hand, may be abstracted as: given access to a string promised to be of the form $x = 0^m 1^{N-m}$, find m.

Classically, oracle access means that one has a black-box subroutine that given i returns x_i. The cost of querying the oracle is taken to be one per query. The hardest inputs to search are clearly those x that are all zeros except in a single position – when there is a single solution – a single "needle in a haystack." (For the OR function, such inputs are hard to distinguish from $x = 0^N$.) For any deterministic search algorithm, there exists such an input on which the algorithm makes at least N oracle queries; brute-force search is the best strategy. Any randomized search algorithm with ε probability of success must make N/ε queries.

Quantumly, one is allowed black-box access to a unitary oracle U_x that can query the oracle in a superposition and get an answer in a superposition. U_x is defined as a controlled reflection about indices i with $x_i = 0$:

$$U_x|c,i\rangle = (-1)^{cx_i}|c,i\rangle \,, \tag{1}$$

where $|c\rangle$ is a control qubit. This can be implemented using U'_x satisfying $U'_x(|c,i,b\rangle) = |c,i,(cx_i)\oplus b\rangle$ – where $b \in \{0,1\}$ and $\oplus$ is addition mod two – by setting the second register to $(1/\sqrt{2})(|0\rangle - |1\rangle)$.

For example, if ϕ is a 3-SAT formula on n variables, $i \in \{1,2,\ldots,N = 2^n\}$ represents a setting for the variables, and x_i indicates if assignment i satisfies ϕ; then is ϕ satisfiable? (Another common example is unstructured database search: i is a record and x_i a function of that record. However, this example is complicated because records need to be stored in a physical memory device. If it is easier to access nearby records, then spatial relationships come into play.)

More generally, say there is a subroutine that returns an efficiently verifiable answer to some problem with probability ε. To solve the problem with constant probability, the subroutine can be run $\Omega(1/\varepsilon)$ times. Quantumly, if the subroutine is a unitary process that returns the right answer with *amplitude* $\sqrt{\varepsilon}$, is there a better technique than measuring the outcome and running the subroutine

$\Omega(1/\varepsilon)$ times? It turns out that this question is closely related to search, because the uniform superposition over indices $(1/\sqrt{N})\sum_i |i\rangle$ has amplitude of returning the right answer as $1/\sqrt{N}$. Thus, an algorithm for this problem immediately implies a search algorithm.

Key Results

Grover [13] showed that there exists a quantum search algorithm that is quadratically faster than the optimal classical randomized algorithm:

Theorem 1 (Grover search) *There is a quantum black-box unstructured search algorithm with success probability ε, using $O(\sqrt{N\varepsilon})$ queries and $O(\sqrt{N\varepsilon}\cdot\log\log N)$ time. If promised that the Hamming weight of x is $|x| \geq k$, then one of the i such that $x_i = 1$ can be found using $O(\sqrt{N\varepsilon/k})$ queries.*

The Grover search algorithm has its simplest form if given the promise that $|x| = 1$. Then the single "marked item" i^* with $x_{i^*} = 1$ can be found by preparing the uniform superposition over indices $|\Psi\rangle = (1/\sqrt{N})\sum_i |i\rangle$, then repeating $\sqrt{N}$ times:

1. Apply the oracle U_x from Eq. (1), with the control bit $c = 1$, to reflect about i^*.
2. Reflect about $|\Psi\rangle$ by applying $U_D = I - 2|\Psi\rangle\langle\Psi|$.

Finally, measure and output i.

It turns out that $i = i^*$ with constant probability. The analysis is straightforward because the quantum state $|\varphi\rangle$ stays in the two-dimensional subspace spanned by $|i^*\rangle$ and $|\Psi\rangle$. Initially, the amplitude on i^* is $\langle i^*|\Psi\rangle = 1/\sqrt{N}$, and the angle between $|i^*\rangle$ and the initial state $|\varphi_0\rangle = |\Psi\rangle$ is $\pi/2 - \theta$, with $\theta = \arcsin 1/\sqrt{N} \approx 1/\sqrt{N}$. Each pair of reflection steps decreases the angle between the $|\varphi\rangle$ and $|i^*\rangle$ by exactly θ, so $\sqrt{N}$ steps suffice to bound the angle away from $\pi/2$. (Using the small angle approximation, after t steps of alternating reflections the amplitude $\langle i^*|\varphi_t\rangle$ is about $t/\sqrt{N}$, making the success probability about t^2/N.)

The reflection about the uniform superposition, $U_D = I - 2|\Psi\rangle\langle\Psi|$, is known as a Grover diffusion step. If the indices are represented in binary, with $N = 2^n$, it can be implemented as transversal Hadamard gates $H^{\otimes n}$, where $H = \frac{1}{\sqrt{2}}\left(\begin{smallmatrix}1 & 1\\ 1 & -1\end{smallmatrix}\right)$), followed by reflection about $|0^n\rangle$, followed by $H^{\otimes n}$ again. This operation can also be interpreted as an inversion about the average of the amplitudes $\{\langle i|\varphi_t\rangle\}$. Note that if one measures i before each query to the oracle, then the algorithm collapses to effectively classical search by random guessing.

Brassard and Høyer [6], and Grover [14] both realized that quantum search can be applied on top of nearly any quantum algorithm for any problem. Roughly, the *amplitude amplification* technique says that given one quantum algorithm that solves a problem with small probability ε, then by using $O(m)$ calls to that algorithm the success probability can be increased to about $m^2\varepsilon$. (Classically, the success probability could only be increased to about $m\varepsilon$.) More formally,

Theorem 2 (Amplitude amplification, [1, Lemma 9]) *Let $\mathcal{A}$ be a quantum algorithm that outputs a correct answer and witness with probability $\delta \leq \varepsilon$ where ε is known. Furthermore, let*

$$m \leq \frac{\pi}{4\arcsin\sqrt{\varepsilon}} - \frac{1}{2}\,.$$

Then there is an algorithm $\mathcal{A}'$ that uses $2m + 1$ calls to $\mathcal{A}$ and $\mathcal{A}^{-1}$ and outputs a correct answer and a witness with probability

$$\delta_{new} \geq \left(1 - \frac{(2m+1)^2}{3}\delta\right)(2m+1)^2\delta\,.$$

Here, one is "searching" for an answer to some problem. The "oracle" is implemented by a routine that conditionally flips the phase based on whether or not the answer is correct (checked using the witness). The reflection about the initial state is implemented by inverting $\mathcal{A}$, applying a reflection about $|0\rangle$, and then reapplying $\mathcal{A}$ (similarly to how the reflection about $|\Psi\rangle$ can be implemented using Hadamard gates). See also [7].

The square-root speedup in quantum search is *optimal*; Bennett, Bernstein, Brassard and Vazirani [4] gave an $\Omega(\sqrt{N})$ lower bound on the number of oracle queries required for a quantum search algorithm. Therefore, quantum computers cannot give an exponential speedup for arbitrary unstructured problems; there exists an oracle relative to which BQP $\not\subseteq$ NP (an NP machine can guess the answer and verify it with one query). In fact, under the promise that $|x| = 1$, the algorithm is precisely optimal and cannot be improved by even a single query [22].

Grover's search algorithm is robust in several ways:

- It is robust against changing both initial state and the diffusion operator:

Theorem 3 ([2]) *Assume $|x| = 1$ with $x_{i^*} = 1$. Assume the initial state $|\varphi_0\rangle$ has real amplitudes $\langle i|\varphi_0\rangle$, with $\langle i^*|\varphi_0\rangle = \alpha$. Let the reflection oracle be $U_x = I - 2|i^*\rangle\langle i^*|$. Let the diffusion operator U_D be a real unitary matrix in the basis $\{|i\rangle\}$. Assume $U_D|\varphi_0\rangle = |\varphi_0\rangle$ and that $U_D|\psi\rangle = e^{i\theta_\psi}|\psi\rangle$ for $\theta_\psi \in [\varepsilon, 2\pi - \varepsilon]$ (where $\varepsilon > 0$ is a constant) for all eigenvectors $|\psi\rangle$ orthogonal to $|\varphi_0\rangle$. Then, there exists $t = O(1/\alpha)$ such that $|\langle i^*|(U_D U_x)^t|\varphi_0\rangle| = \Omega(1)$. (The constant under $\Omega(1)$ is independent of α but can depend on ε.)*

Therefore, there is in fact an entire class of related algorithms that use different "diffusion" operators. This robustness is useful in applications, and may help to explain why Grover search ideas appear so frequently in quantum algorithms.

- Høyer, Mosca and de Wolf [16] showed that quantum search can be implemented so as to be robust also against faulty oracles, a problem known as Bounded-Error Search:

Theorem 4 *Suppose* $U''_x|i,b\rangle = \sqrt{p_i}|i, x_i \oplus b\rangle + \sqrt{1-p_i}|i,(1-x_i)\oplus b\rangle$, *with each* $p_i \geq 9/10$ *(i. e., there is a bounded coherent error rate in the oracle). Search can still be implemented in* $O(\sqrt{N})$ *time.*

Applications

An early application of the Grover search algorithm was to finding collisions; given oracle access to a 2-to-1 function f, find distinct points x, y such that $f(x) = f(y)$. Brassard, Høyer and Tapp [8] developed an $O(N^{1/3})$-time collision algorithm. Finding $x \neq y$ such that $f(x) = f(y)$ for a general function f is known as the Element-Distinctness problem. Buhrman et al. [9] later developed an $O(N^{3/4}\log N)$-time algorithm for Element Distinctness, using amplitude amplification. In a breakthrough, Ambainis [2] gave an optimal, $O(N^{2/3})$-time algorithm for Element Distinctness, which has also led to other applications [18]. Ambainis's algorithm extends quantum search by using a certain quantum walk to replace the Grover diffusion step, and uses Theorem 3 for its analysis.

Grover search has also proved useful in communication complexity. For example, a straightforward distributed implementation of the search algorithm solves the Set Intersection problem – Alice and Bob have respective inputs $x, y \in \{0,1\}^N$, and want to find an index i such that $x_i = y_i = 1$ – with $O(\sqrt{N}\log N)$ qubits of communication. Recently, this technique has led to an exponential classical/quantum separation in the memory required to evaluate a certain total function with a streaming input [17].

Unlike the usual Grover search that has an oscillatory behavior, fixed-point quantum search algorithms converge monotonically to the solution. These algorithms replace the reflection operation – a phase shift of π – with selective phase shifts of $\pi/3$.

Theorem 5 ([15]) *Let* R_s *and* R_t *be selective* $\pi/3$ *phase shifts of the source and target state(s), respectively. If* $\|\langle t|U\rangle|s\rangle\|^2 = 1-\varepsilon$, *then* $\left\|\langle t|\, UR_sU^\dagger R_tU\,|s\rangle\right\|^2 = 1-\varepsilon^3$.

In other words, the deviation of the final state from the desired final state reduces to the cube of what it was for the original transformation. (Classically only an $O(\varepsilon^2)$ improvement is possible.) This clearly gives a monotonic improvement towards the solution state, which is useful when the number of solutions is very high. The technique has also been applied to composite pulses [19]. However, it does not give a square-root speedup.

Another extension of unstructured search is to game-tree evaluation, which is a recursive search problem. Classically, using the alpha-beta pruning technique, the value of a balanced binary AND-OR tree can be computed with zero error in expected time $O(N^{\log_2[(1+\sqrt{33})/4]}) = O(N^{0.754})$[20], and this is optimal even for bounded-error algorithms [21]. Applying quantum search recursively, a depth-d regular AND-OR tree can be evaluated with constant error in time $\sqrt{N}\cdot O(\log N)^{d-1}$, where the log factors come from amplifying the success probability of inner searches to be close to one. Bounded-error quantum search, Theorem 4, allows eliminating these log factors, so the time becomes $O(\sqrt{N}\cdot c^d)$, for some constant c. Very recently, an $N^{1/2+o(1)}$-time algorithm has been discovered for evaluating an arbitrary AND-OR tree on N variables [3,11,12].

Open Problems

As already mentioned, search of a sorted list may be abstracted as, given $x = 0^m1^{N-m}$, find m. Classically, $\lceil\log_2 N\rceil$ queries are necessary and sufficient to find m, with binary search achieving the optimum. Quantumly, $\Theta(\log N)$ queries are also necessary and sufficient, but the constant is unknown. The best lower bound on an exact algorithm (i. e., which succeeds with probability one after a fixed number of queries) is about $0.221\log_2 N$ and the best exact algorithm uses about $0.443\log_2 N$ queries (although there is a quantum Las Vegas algorithm that uses expected $0.32\log_2 N$ queries) [10].

Cross References

▶ Quantum Algorithm for Element Distinctness
▶ Routing

Recommended Reading

1. Aaronson, S., Ambainis A.: Quantum search of spatial regions. Theor. Comput. **1**, 47–79 (2005)
2. Ambainis, A.: Quantum walk algorithm for element distinctness. SIAM J. Comput. **37**(1), 210–239 (2007)
3. Ambainis, A.: A nearly optimal discrete query quantum algorithm for evaluating NAND formulas, arXiv:0704.3628 (2007)
4. Bennett, C.H., Bernstein, E., Brassard, G., Vazirani, U.: Strengths and weaknesses of quantum computing. SIAM J. Comput. **26**(5), 1510–1523 (1997)

5. Brassard, G.: Searching a quantum phone book. Science **275**(5300), 627–628 (1997)
6. Brassard, G., Høyer, P.: An exact quantum polynomial-time algorithm for Simon's problem. In: Proc. 5th Israeli Symp. on Theory of Computing and Systems (ISTCS), pp. 12–23. IEEE Computer Society Press, Hoboken (1997)
7. Brassard, G., Høyer, P., Mosca, M., Tapp, A.: Quantum amplitude amplification and estimation. In: Quantum Computation and Quantum Information Science. AMS Contemporary Mathematics Series, vol. 305 Contemporary Mathematics, pp. 53–74, Providence (2002)
8. Brassard, G., Høyer, P., Tapp, A.: Quantum cryptanalysis of hash and claw-free functions. In: Proc. 3rd Latin American Theoretical Informatics Conference (LATIN). Lecture Notes in Computer Science, vol. 1380, pp. 163–169. Springer, New York (1998)
9. Buhrman, H., Dürr, C., Heiligman, M., Høyer, P., Magniez, F., Santha, M., de Wolf, R. Quantum algorithms for element distinctness, quant-ph/0007016 (2000)
10. Childs, A.M., Landahl A.J., Parrilo, P.A.: Improved quantum algorithms for the ordered search problem via semidefinite programming. Phys. Rev. A **75**, 032335 (2007)
11. Childs, A.M., Reichardt, B.W., Špalek, R., Zhang, S.: Every NAND formula of size N can be evaluated in time $N^{1/2+o(1)}$ on a quantum computer, quant-ph/0703015 (2007)
12. Farhi, E., Goldstone, J., Gutmann, S.: A quantum algorithm for the Hamiltonian NAND tree. quant-ph/0702144 (2007)
13. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: Proc. 28th ACM Symp. on Theory of Computing (STOC), pp. 212–219. Philadelphia, 22–24 May 1996
14. Grover, L.K.: A framework for fast quantum mechanical algorithms. In: Proc. 30th ACM Symp. on Theory of Computing (STOC), pp. 53–62. Dallas, 23–26 May 1998
15. Grover, L.K.: Fixed-point quantum search. Phys. Rev. Lett. **95**, 150501 (2005)
16. Høyer, P., Mosca, M., de Wolf, R.: Quantum search on bounded-error inputs. In: Proc. 30th International Colloquium on Automata, Languages and Programming (ICALP 03), Eindhoven, The Netherlands, pp. 291–299 (2003)
17. Le Gall, F.: Exponential separation of quantum and classical online space complexity. In: Proc. ACM Symp. on Parallel Algorithms and Architectures (SPAA), Cambride, 30 July–1 August (2006)
18. Magniez, F., Nayak, A., Roland, J., Santha, M.: Search via quantum walk. quant-ph/0608026. In: Proc. of 39th ACM Symp. on Theory of Computing (STOC), San Diego, 11–13 June, pp. 575–584 (2007)
19. Reichardt, B.W., Grover, L.K.: Quantum error correction of systematic errors using a quantum search framework. Phys. Rev. A **72**, 042326 (2005)
20. Saks, M., Wigderson, A.: Probabilistic Boolean decision trees and the complexity of evaluating game trees. In: Proc. of 27th IEEE Symp. on Foundation of Computer Science (FOCS), Toronto, 27–29 October, pp. 29–38 (1986)
21. Santha, M.: On the Monte Carlo decision tree complexity of read-once formulae. Random Struct. Algorit. **6**(1), 75–87 (1995)
22. Zalka, C.: Grover's quantum searching algorithm is optimal. Phys. Rev. A **60**(4), 2746–2751 (1999)

Quickest Route

► All Pairs Shortest Paths in Sparse Graphs
► Single-Source Shortest Paths

Quorums

1985; Garcia-Molina, Barbara

DAHLIA MALKHI
Microsoft, Silicon Valley Campus,
Mountain View, CA, USA

Keywords and Synonyms

Quorum systems; Voting systems; Coteries

Problem Definition

Quorum systems are tools for increasing the availability and efficiency of replicated services. A *quorum system* for a universe of servers is a collection of subsets of servers, each pair of which intersect. Intuitively, each quorum can operate on behalf of the system, thus increasing its availability and performance, while the intersection property guarantees that operations done on distinct quorums preserve consistency.

The motivation for quorum systems stems from the need to make critical missions performed by machines that are reliable. The only way to increase the reliability of a service, aside from using intrinsically more robust hardware, is via replication. To make a service robust, it can be installed on multiple identical servers, each one of which holds a copy of the service state and performs read/write operations on it. This allows the system to provide information and perform operations even if some machines fail or communication links go down. Unfortunately, replication incurs a cost in the need to maintain the servers consistent. To enhance the availability and performance of a replicated service, Gifford and Thomas introduced in 1979 [3,14] the usage of *votes* assigned to each server, such that a majority of the sum of votes is sufficient to perform operations. More generally, quorum systems are defined formally as follows:

Quorum system: Assume a *universe* U of servers, $|U| = n$, and an arbitrary number of clients. A *quorum system* $\mathcal{Q} \subseteq 2^U$ is a set of subsets of U, every pair of which intersect. Each $Q \in \mathcal{Q}$ is called a *quorum*.

Access Protocol

To demonstrate the usability of quorum systems in constructing replicated services, quorums are used here to implement a multi-writer multi-reader atomic shared variable. Quorums have also been used in various *mutual exclusion* protocols, to achieve Consensus, and in commit protocols.

In the application, clients perform read and write operations on a variable x that is replicated at each server in the universe U. A copy of the variable x is stored at each server, along with a timestamp value t. Timestamps are assigned by a client to each replica of the variable when the client writes the replica. Different clients choose distinct timestamps, e. g., by choosing integers appended with the name of c in the low-order bits. The read and write operations are implemented as follows.

Write: For a client c to write the value v, it queries each server in some quorum Q to obtain a set of value/timestamp pairs $A = \{\langle v_u, t_u \rangle\}_{u \in Q}$; chooses a timestamp $t \in T_c$ greater than the highest timestamp value in A; and updates x and the associated timestamp at each server in Q to v and t, respectively.

Read: For a client to read x, it queries each server in some quorum Q to obtain a set of value/timestamp pairs $A = \{\langle v_u, t_u \rangle\}_{u \in Q}$. The client then chooses the pair $\langle v, t \rangle$ with the highest timestamp in A to obtain the result of the read operation. It writes back $\langle v, t \rangle$ to each server in some quorum Q'.

In both read and write operations, each server updates its local variable and timestamp to the received values $\langle v, t \rangle$ only if t is greater than the timestamp currently associated with the variable. The above protocol correctly implements the semantics of a multi-writer multi-reader atomic variable (see ▶ Linearizability).

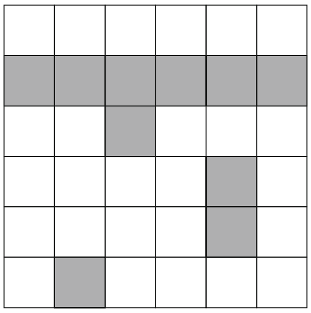
Key Results

Perhaps the two most obvious quorum systems are the singleton, and the set of majorities, or more generally, weighted majorities suggested by Gifford [3].

Singleton: The set system $\mathcal{Q} = \{\{u\}\}$ for some $u \in U$ is the singleton quorum system.

Weighted Majorities: Assume that every server s in the universe U is assigned a number of votes w_s. Then, the set system $\mathcal{Q} = \{Q \subseteq U : \sum_{q \in Q} w_q > (\sum_{q \in U} w_q)/2\}$ is a quorum system called Weighted Majorities. When all the weights are the same, simply call this the system of Majorities.

An example of a quorum system that cannot be defined by voting is the following Grid construction:



Quorums, Figure 1
The Grid quorum system of 6 × 6, with one quorum shaded

Grid: Suppose that the universe of servers is of size $n = k^2$ for some integer k. Arrange the universe into a $\sqrt{n} \times \sqrt{n}$ grid, as shown in Fig. 1. A quorum is the union of a full row and one element from each row below the full row. This yields the Grid quorum system, whose quorums are of size $O(\sqrt{n})$.

Maekawa suggests in [6] a quorum system that has several desirable symmetry properties, and in particular, that every pair of quorums intersect in exactly one element:

FPP: Suppose that the universe of servers is of size $n = q^2 + q + 1$, where $q = p^r$ for a prime p. It is known that a finite projective plane exists for n, with $q + 1$ pairwise intersecting subsets, each subset of size $q + 1$, and where each element is contained in $q + 1$ subsets. Then the set of finite projective plane subsets forms a quorum system.

Voting and Related Notions

Since generally it would be senseless to access a large quorum if a subset of it is a quorum, a good definition may avoid such anomalies. Garcia-Molina and Barbara [2] call such well-formed systems *coteries*, defined as follows:

Coterie: A *coterie* $\mathcal{Q} \subseteq 2^U$ is a quorum system such that for any $Q, Q' \in \mathcal{Q} : Q \not\subseteq Q'$.

Of special interest are quorum systems that cannot be reduced in size (i. e., that no quorum in the system can be reduced in size). Garcia-Molina and Barbara [2] use the term "dominates" to mean that one quorum system is always superior to another, as follows:

Domination: Suppose that $\mathcal{Q}, \mathcal{Q}'$ are two coteries, $\mathcal{Q} \neq \mathcal{Q}'$, such that for every $Q' \in \mathcal{Q}'$, there exists a $Q \in \mathcal{Q}$ such that $Q \subseteq Q'$. Then $\mathcal{Q}$ *dominates* $\mathcal{Q}'$.$\mathcal{Q}'$ is *dominated* if there exists a coterie $\mathcal{Q}$ that dominates it, and is *non-dominated* if no such coterie exists.

Voting was mentioned above as an intuitive way of thinking about quorum techniques. As it turns out, vote assignments and quorums are not equivalent. Garcia-

Molina and Barbara [2] show that quorum systems are strictly more general than voting, i. e. each vote assignment has some corresponding quorum system but not the other way around. In fact, for a system with n servers, there is a double-exponential ($2^{2^{cn}}$) number of non-dominated coteries, and only $O(2^{n^2})$ different vote assignments, though for $n \leq 5$, voting and non-dominated coteries are identical.

Measures

Several measures of quality have been identified to address the question of which quorum system works best for a given set of servers; among these, *load* and *availability* are elaborated on here.

Load A measure of the inherent performance of a quorum system is its *load*. Naor and Wool define in [10] the load of a quorum system as the probability of accessing the busiest server in the *best* case. More precisely, given a quorum system $\mathcal{Q}$, an *access strategy* w is a probability distribution on the elements of $\mathcal{Q}$; i. e., $\sum_{Q \in \mathcal{Q}} w(Q) = 1$. $w(Q)$ is the probability that quorum Q will be chosen when the service is accessed. Load is then defined as follows:

Load: Let a strategy w be given for a quorum system $\mathcal{Q} = \{Q_1, \ldots, Q_m\}$ over a universe U. For an element $u \in U$, the load induced by w on u is $l_w(u) = \sum_{Q_i \ni u} w(Q_i)$. The load induced by a strategy w on a quorum system $\mathcal{Q}$ is

$$L_w(\mathcal{Q}) = \max_{u \in U}\{l_w(u)\}.$$

The *system load* (or just *load*) on a quorum system $\mathcal{Q}$ is

$$L(\mathcal{Q}) = \min_{w}\{L_w(\mathcal{Q})\},$$

where the minimum is taken over all strategies.

The load is a best-case definition, and will be achieved only if an optimal access strategy is used, and only in the case that no failures occur. A strength of this definition is that load is a property of a quorum system, and not of the protocol using it.

The following theorem was proved in [10] for all quorum systems.

Theorem 1 *Let $\mathcal{Q}$ be a quorum system over a universe of n elements. Denote by $c(\mathcal{Q})$ the size of the smallest quorum of $\mathcal{Q}$. Then $L(\mathcal{Q}) \geq \max\{\frac{1}{c(\mathcal{Q})}, \frac{c(\mathcal{Q})}{n}\}$. Consequently, $L(\mathcal{Q}) \geq \frac{1}{\sqrt{n}}$.*

Availability The resilience f of a quorum system provides one measure of how many crash failures a quorum system is *guaranteed* to survive.

Resilience: The *resilience* f of a quorum system $\mathcal{Q}$ is the largest k such that for every set $K \subseteq U$, $|K| = k$, there exists $Q \in \mathcal{Q}$ such that $K \cap Q = \emptyset$.

Note that, the resilience f is at most $c(\mathcal{Q}) - 1$, since by disabling the members of the smallest quorum every quorum is hit. It is possible, however, that an f-resilient quorum system, though vulnerable to a few failure configurations of $f + 1$ failures, can survive many configurations of more than f failures. One way to measure this property of a quorum system is to assume that each server crashes independently with probability p and then to determine the probability F_p that no quorum remains completely alive. This is known as *failure probability* and is formally defined as follows:

Failure probability: Assume that each server in the system crashes independently with probability p. For every quorum $Q \in \mathcal{Q}$ let $\mathcal{E}_Q$ be the event that Q is *hit*, i. e., at least one element $i \in Q$ has crashed. Let crash($\mathcal{Q}$) be the event that all the quorums $Q \in \mathcal{Q}$ were hit, i. e., $\text{crash}(\mathcal{Q}) = \bigwedge_{Q \in \mathcal{Q}} \mathcal{E}_Q$. Then the system failure probability is $F_p(\mathcal{Q}) = \Pr(\text{crash}(\mathcal{Q}))$.

Peleg and Wool study the availability of quorum systems in [11]. A good failure probability $F_p(\mathcal{Q})$ for a quorum system $\mathcal{Q}$ has $\lim_{n \to \infty} F_p(\mathcal{Q}) = 0$ when $p < \frac{1}{2}$. Note that, the failure probability of any quorum system whose resilience is f is at least $e^{-\Omega(f)}$. Majorities has the best availability when $p < \frac{1}{2}$; for $p = \frac{1}{2}$, there exist quorum constructions with $F_p(\mathcal{Q}) = \frac{1}{2}$; for $p > \frac{1}{2}$, the singleton has the best failure probability $F_p(\mathcal{Q}) = p$, but for most quorum systems, $F_p(\mathcal{Q})$ tends to 1.

The Load and Availability of Quorum Systems

Quorum constructions can be compared by analyzing their behavior according to the above measures. The singleton has a load of 1, resilience 0, and failure probability $F_p = p$. This system has the best failure probability when $p > \frac{1}{2}$, but otherwise performs poorly in both availability and load.

The system of Majorities has a load of $\lceil \frac{n+1}{2n} \rceil \approx \frac{1}{2}$. It is resilient to $\lfloor \frac{n-1}{2} \rfloor$ failures, and its failure probability is $e^{-\Omega(n)}$. This system has the highest possible resilience and asymptotically optimal failure probability, but poor load.

Grid's load is $O(\frac{1}{\sqrt{n}})$, which is within a constant factor from optimal. However, its resilience is only $\sqrt{n} - 1$ and it has poor failure probability which tends to 1 as n grows.

The resilience of a FPP quorum system is $q \approx \sqrt{n}$. The load of FPP was analyzed in [10] and shown to be $L(\text{FPP}) = \frac{q+1}{n} \approx 1/\sqrt{n}$, which is optimal. However, its failure probability tends to 1 as n grows.

As demonstrated by these systems, there is a tradeoff between load and fault tolerance in quorum systems, where the resilience f of a quorum system $\mathcal{Q}$ satisfies $f \leq nL(\mathcal{Q})$. Thus, improving one must come at the expense of the other, and it is in fact impossible to simultaneously achieve both optimally. One might conclude that good load conflicts with low failure probability, which is not necessarily the case. In fact, there exist quorum systems such as the Paths system of Naor and Wool [10] and the Triangle Lattice of Bazzi [1] that achieve asymptotically optimal load of $O(1/\sqrt{n})$ and have close to optimal failure probability for their quorum sizes. Another construction is the CWlog system of Peleg and Wool [12], which has unusually small quorum sizes of $\log n - \log\log n$, and for systems with quorums of this size, has optimal load, $L(\text{CWlog}) = O(1/\log n)$, and optimal failure probability.

Byzantine Quorum Systems

For the most part, quorum systems were studied in environments where failures may simply cause servers to become unavailable (benign failures). But what if a server may exhibit arbitrary, possibly malicious behavior? Malkhi and Reiter [7] carried out a study of quorum systems in environments prone to arbitrary (Byzantine) behavior of servers. Intuitively, a quorum system tolerant of Byzantine failures is a collection of subsets of servers, each pair of which intersect in a set containing sufficiently many *correct* servers to mask out the behavior of faulty servers. More precisely, Byzantine quorum systems are defined as follows:

Masking quorum system: A quorum system $\mathcal{Q}$ is a *b-masking quorum system* if it has resilience $f \geq b$, and each pair of quorums intersect in at least $2b + 1$ elements.

The masking quorum system requirements enable a client to obtain the correct answer from the service despite up to b Byzantine server failures. More precisely, a write operation remains as before; to obtain the correct value of x from a read operation, the client reads a set of value/timestamp pairs from a quorum Q and sorts them into clusters of identical pairs. It then chooses a value/timestamp pair that is returned from at least $b + 1$ servers, and therefore must contain at least one correct server. The properties of masking quorum systems guarantee that at least one such cluster exists. If more than one such cluster exists, the client chooses the one with the highest timestamp. It is easy to see that any value so obtained was written before, and moreover, that the most recently written value is obtained. Thus, the semantics of a multi-writer multi-reader safe variable are obtained (see ► Linearizability) in a Byzantine environment.

For a b-masking quorum system, the following lower bound on the load holds:

Theorem 2 *Let $\mathcal{Q}$ be a b-masking quorum system. Then $L(\mathcal{Q}) \geq \max\{\frac{2b+1}{c(\mathcal{Q})}, \frac{c(\mathcal{Q})}{n}\}$, and consequently $L(\mathcal{Q}) \geq \sqrt{\frac{2b+1}{n}}$.*

This bound is tight, and masking quorum constructions meeting it were shown.

Malkhi and Reiter explore in [7] two variations of masking quorum systems. The first, called *dissemination quorum systems*, is suited for services that receive and distribute *self-verifying* information from correct clients (e. g., digitally signed values) that faulty servers can fail to redistribute but cannot undetectably alter. The second variation, called *opaque masking quorum systems*, is similar to regular masking quorums in that it makes no assumption of self-verifying data, but it differs in that clients do not need to know the failure scenarios for which the service was designed. This somewhat simplifies the client protocol and, in the case that the failures are maliciously induced, reveals less information to clients that could guide an attack attempting to compromise the system. It is also shown in [7] how to deal with faulty clients in addition to faulty servers.

Probabilistic Quorum Systems

The resilience of any quorum system is bounded by half of the number of servers. Moreover, as mentioned above, there is an inherent tradeoff between low load and good resilience, so that it is in fact impossible to simultaneously achieve both optimally. In particular, quorum systems over n servers that achieve the optimal load of $\frac{1}{\sqrt{n}}$ can tolerate at most $\sqrt{n}$ faults.

To break these limitations, Malkhi et al. propose in [8] to relax the intersection property of a quorum system so that "quorums" chosen according to a specified strategy intersect only with very high probability. They accordingly name these *probabilistic quorum systems*. These systems admit the possibility, albeit small, that two operations will be performed at non-intersecting quorums, in which case consistency of the system may suffer. However, even a small relaxation of consistency can yield dramatic improvements in the resilience and failure probability of the system, while the load remains essentially unchanged. Probabilistic quorum systems are thus most suitable for use when availability of operations despite the presence of faults is more important than certain consistency. This might be the case if the cost of inconsistent operations is high but not irrecoverable, or if obtaining the most up-to-date information is desirable but not critical, while having no information may have heavier penalties.

The family of constructions suggested in [8] is as follows:

$\mathbf{W(n, \ell)}$ Let U be a universe of size n.$W(n, \ell)$, $\ell \geq 1$, is the system $\langle \mathcal{Q}, w \rangle$ where $\mathcal{Q}$ is the set system $\mathcal{Q} = \{Q \subseteq U : |Q| = \ell\sqrt{n}\}$; w is an access strategy w defined by $\forall Q \in \mathcal{Q}, w(Q) = \frac{1}{|\mathcal{Q}|}$.

The probability of choosing according to w two quorums that do not intersect is less than $e^{-\ell^2}$, and can be made sufficiently small by appropriate choice of ℓ. Since every element is in $\binom{n-1}{\ell\sqrt{n}-1}$ quorums, the load $L(W(n, \ell))$ is $\frac{\ell}{\sqrt{n}} = O(\frac{1}{\sqrt{n}})$. Because only $\ell\sqrt{n}$ servers need be available in order for some quorum to be available, $W(n, \ell)$ is resilient to $n - \ell\sqrt{n}$ crashes. The failure probability of $W(n, \ell)$ is less than $e^{-\Omega(n)}$ for all $p \leq 1 - \frac{\ell}{\sqrt{n}}$, which is asymptotically optimal. Moreover, if $\frac{1}{2} \leq p \leq 1 - \frac{\ell}{\sqrt{n}}$, this probability is provably better than any (non-probabilistic) quorum system.

Relaxing consistency can also provide dramatic improvements in environments that may experience Byzantine failures. More details can be found in [8].

Applications

Just about any fault tolerant distributed protocol, such as Paxos [5] or consensus [1] implicitly builds on quorums, typically majorities. More concretely, scalable data repositories were built, such as Fleet [9], Rambo [4], and Rosebud [13].

Cross References

► Concurrent Programming, Mutual Exclusion

Recommended Reading

1. Dwork, C., Lynch, N., Stockmeyer, L.: Consensus in the presence of partial synchrony. J. Assoc. Comput. Mach. **35**, 288–323 (1988)
2. Garcia-Molina, H., Barbara, D.: How to assign votes in a distributed system. J. ACM **32**, 841–860 (1985)
3. Gifford, D.K.: Weighted voting for replicated data. In: Proceedings of the 7th ACM Symposium on Operating Systems Principles, 1979, pp. 150–162
4. Gilbert, S.: Lynch, N., Shvartsman, A., Rambo ii: Rapidly reconfigurable atomic memory for dynamic networks. pp. 259–268. In: Proceedings if the IEEE 2003 International Conference on Dependable Systems and Networks (DNS). San Francisco, USA (2003)
5. Lamport, L.: The part-time parliament. ACM Trans. Comput. Syst. **16**, 133–169 (1998)
6. Maekawa, M.: A $\sqrt{n}$ algorithm for mutual exclusion in decentralized systems. ACM Trans. Comput. Syst. **3**(2), 145–159 (1985)
7. Malkhi, D., Reiter, M.: Byzantine quorum systems. Distrib. Comput. **11**, 203–213 (1998)
8. Malkhi, D., Reiter, M., Wool, A., Wright, R.: Probabilistic quorum systems. Inf. Comput. J. **170**, 184–206 (2001)
9. Malkhi, D., Reiter, M.K.: An architecture for survivable coordination in large-scale systems. IEEE Trans. Knowl. Data Engineer. **12**, 187–202 (2000)
10. Naor, M., Wool, A.: The load, capacity and availability of quorum systems. SIAM J. Comput. **27**, 423–447 (1998)
11. Peleg, D., Wool, A.: The availability of quorum systems. Inf. Comput. **123**, 210–223 (1995)
12. Peleg, D., Wool, A.: Crumbling walls: A class of practical and efficient quorum systems. Distrib. Comput. **10**, 87–98 (1997)
13. Rodrigues, R., Liskov, B.: Rosebud: A scalable byzantine-fault tolerant storage architecture. In: Proceedings of the 18th ACM Symposium on Operating System Principles, San Francisco, USA (2003)
14. Thomas, R.H.: A majority consensus approach to concurrency control for multiple copy databases. ACM Trans. Database Syst. **4**, 180–209 (1979)

R

Radiocoloring in Planar Graphs

2005; Fotakis, Nikoletseas, Papadopoulou, Spirakis

VICKY PAPADOPOULOU
Department of Computer Science, University of Cyprus, Nicosia, Cyprus

Keywords and Synonyms

λ-coloring; k-coloring; Distance-2 coloring; Coloring the square of the graph

Problem Definition

Consider a graph $G(V, E)$. For any two vertices $u, v \in V$, $d(u, v)$ denotes the distance of u, v in G. The general problem concerns a coloring of the graph G and it is defined as follows:

Definition 1 (*k*-coloring problem)
INPUT: A graph $G(V, E)$.
OUTPUT: A function $\phi : V \rightarrow \{1, \dots, \infty\}$, called *k-coloring of* G such that $\forall u, v \in V$, $x \in \{0, 1, \dots, k\}$: if $d(u, v) \geq k - x + 1$ then $|\phi(u) - \phi(v)| = x$.
OBJECTIVE: Let $|\phi(V)| = \lambda_\phi$. Then λ_ϕ is the *number of colors* that φ actually uses (it is usually called *order* of G under φ). The number $\nu_\phi = max_{v \in V} \phi(v) - min_{u \in V} \phi(u) + 1$ is usually called the *span* of G under φ. The function φ satisfies one of the following objectives:

- minimum span: λ_ϕ is the minimum possible over all possible functions φ of G;
- minimum order: ν_ϕ is the minimum possible over all possible functions φ of G;
- Min span order: obtains a minimum span and moreover, from all minimum span assignments, φ obtains a minimum order.
- Min order span: obtains a minimum order and moreover, from all minimum order assignments, φ obtains a minimum span.

Note that the case $k = 1$ corresponds to the well known problem of *vertex graph coloring*. Thus, k-coloring problem (with k as an input) is $\mathcal{NP}$-complete [4]. The case of k-coloring problem where $k = 2$, is called the *Radiocoloring problem*.

Definition 2 (Radiocoloring Problem (RCP) [7])
INPUT: A graph $G(V, E)$.
OUTPUT: A function $\Phi : V \rightarrow N^*$ such that $|\Phi(u) - \Phi(v)| \geq 2$ if $d(u, v) = 1$ and $|\Phi(u) - \Phi(v)| \geq 1$ if $d(u, v) = 2$.
OBJECTIVE: The least possible number (order) needed to radiocolor G is denoted by $X_{\text{order}}(G)$. The least possible number $\max_{v \in V} \Phi(v) - \min_{u \in V} \Phi(u) + 1$ (span) needed for the radiocoloring of G is denoted as $X_{\text{span}}(G)$. Function Φ satisfies one of the followings:

- Min span RCP: Φ obtains a minimum span, i. e. $\lambda_\Phi = X_{span}(G)$;
- Min order RCP: Φ obtains a minimum order $\nu_\Phi = X_{\text{order}}(G)$;
- Min span order RCP: obtains a minimum span and moreover, from all minimum span assignments, Φ obtains a minimum order.
- Min order span RCP: obtains a minimum order and moreover, from all minimum order assignments, Φ obtains a minimum span.

A related to the RCP problem concerns to the square of a graph G, which is defined as follows:

Definition 3 Given a graph $G(V, E)$, G^2 is the graph having the same vertex set V and an edge set $E' : \{u, v\} \in E'$ iff $d(u, v) \leq 2$ in G.

The related problem is to color the square of a graph G, G^2 so that no two neighbor vertices (in G^2) get the same color. The objective is to use a minimum number of colors, denoted as $\chi(G^2)$ and called *chromatic number of the square of the graph* G. [5,6] first observed that for any graph G, $X_{\text{order}}(G)$ is the same as the (vertex) chromatic number of G^2, i. e. $X_{\text{order}}(G) = \chi(G^2)$.

Key Results

[5,6] studied *min span order*, *min order* and *min span* RCP in *planar* graph G. A planar graph, is a graph for which its edges can be embedded in the plane without crossings. The following results are obtained:

- It is first shown that the number of colors used in the *min span order RCP* of graph G is different from the chromatic number of the square of the graph, $\chi(G^2)$. In particular, it may be greater than $\chi(G^2)$.
- It is then proved that the radiocoloring problem for general graphs is hard to approximate (unless $\mathcal{NP} = ZPP$, the class of problems with polynomial time zero-error randomized algorithms) within a factor of $n^{1/2-\epsilon}$ (for any $\epsilon > 0$), where n is the number of vertices of the graph. However, when restricted to some special cases of graphs, the problem becomes easier.
  It is shown that the *min span RCP* and *min span order RCP* are $\mathcal{NP}$*-complete* for planar graphs. Note that few combinatorial problems remain hard for *planar* graphs and their proofs of hardness are not easy since they have to use planar gadgets which are difficult to find and understand.
- It presents a $O(n\Delta(G))$ time algorithm that *approximates* the min order of RCP, X_{order}, of a planar graph G *by a constant ratio which tends to 2* as the maximum degree $\Delta(G)$ of G increases.
  The algorithm presented is motivated by a constructive coloring theorem of Heuvel and McGuiness [9]. The construction of [9] can lead (as shown) to an $O(n^2)$ technique assuming that a planar embedding of G is given. [5,6] improves the time complexity of the approximation, and presents a much more simple algorithm to verify and implement. The algorithm does not need any planar embedding as input.
- Finally, the work considers the problem of *estimating the number of different radiocolorings* of a planar graph G. This is a $\#\mathcal{P}$-complete problem (as can be easily seen from the completeness reduction presented there that can be done parsimonious). They authors employ here standard techniques of rapidly mixing Markov Chains and the *new method of coupling* for purposes of proving *rapid convergence* (see e. g. [10]) and present *a fully polynomial randomized approximation scheme* for estimating the number of radiocolorings with λ colors for a planar graph G, when $\lambda \geq 4\Delta(G) + 50$.

In [8] and [7] it has been proved that the problem of min span RCP is $\mathcal{NP}$-complete, even for graphs of diameter 2. The reductions use highly non-planar graphs. In [11] it is proved that the problem of coloring the square of a general graph is $\mathcal{NP}$-complete.

Another variation of RCP for planar graphs, called *distance-2-coloring* is studied in [12]. This is the problem of coloring a given graph G with the minimum number of colors so that the vertices of distance *at most* two get different colors. Note that this problem is equivalent to coloring the square of the graph G, G^2. In [12] it is proved that the distance-2-coloring problem for planar graphs is $\mathcal{NP}$-complete. As it is shown in [5,6], this problem is different from the min span order RCP. Thus, the $\mathcal{NP}$-completeness proof in [12] certainly does not imply the $\mathcal{NP}$-completeness of min span order RCP proved in [5,6]. In [12] a 9-approximation algorithm for the distance-2-coloring of planar graphs is also provided.

Independently and in parallel, Agnarsson and Halldórsson in [1] presented approximations for the chromatic number of square and power graphs (G^k). In particular they presented an 1.8-approximation algorithm for coloring the square of a planar graph of large degree ($\Delta(G) \geq 749$). Their method utilizes the notion of *inductiveness* of the square of a planar graph.

Bodlaender et al. in [2] proved also independently and and in parallel that the min span RCP, called λ-labeling there, is $\mathcal{NP}$-complete for planar graphs, using a similar to the approach used in [5,6]. In the same work the authors presented approximations for the problem for some interesting families of graphs: outerplanar graphs, graphs of bounded treewidth, permutation and split graphs.

Applications

The Frequency Assignment Problem (FAP) in radio networks is a well-studied, interesting problem, aiming at assigning frequencies to transmitters exploiting frequency reuse while keeping signal interference to acceptable levels. The interference between transmitters are modeled by an interference graph $G(V, E)$, where V ($|V| = n$) corresponds to the set of transmitters and E represents distance constraints (e. g. if two neighbor nodes in G get the same or close frequencies then this causes unacceptable levels of interference). In most real life cases the network topology formed has some special properties, e. g. G is a lattice network or a planar graph. Planar graphs are mainly the object of study in [5,6].

The FAP is usually modeled by variations of the graph coloring problem. The set of colors represents the available frequencies. In addition, each color in a particular assignment gets an integer value which has to satisfy certain inequalities compared to the values of colors of nearby nodes in G (frequency-distance constraints). A discrete version of FAP is the k-coloring problem, of which a particular instance, for $k = 2$, is investigated in [5,6].

Real networks reserve bandwidth (range of frequencies) rather than distinct frequencies. In this case, an assignment seeks to use as small range of frequencies as possible. It is sometimes desirable to use as few distinct frequencies of a given bandwidth (span) as possible, since the unused frequencies are available for other use. However, there are cases where the primary objective is to minimize the number of frequencies used and the span is a secondary objective, since we wish to avoid reserving unnecessary large span. These realistic scenaria directed researchers to consider optimization versions of the RCP, where one aims in minimizing the span (bandwidth) or the order (distinct frequencies used) of the assignment. Such optimization problems are investigated in [5,6].

Cross References

► Channel Assignment and Routing in Multi-Radio Wireless Mesh Networks
► Graph Coloring

Recommended Reading

1. Agnarsson, G., Halldórsson, M.M.: Coloring Powers of Planar Graphs. In: Proceedings of the 11th Annual ACM-SIAM symposium on Discrete algorithms, pp. 654–662 (2000)
2. Bodlaender, H.L., Kloks, T., Tan, R.B., van Leeuwen, J.: Approximations for λ-Coloring of Graphs. In: Proceedings of the 17th Annual Symposium on Theoretical Aspects of Computer Science. Lecture Notes in Computer Science, vol. 1770, pp. 395-406. Springer (2000)
3. Hale, W.K.: Frequency Assignment: Theory and Applications. In: Proceedings of the IEEE, vol. 68, number 12, pp. 1497-1514 (1980)
4. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of $\mathcal{NP}$-Completeness, W.H. Freeman and Co. (1979)
5. Fotakis, D., Nikoletseas, S., Papadopoulou, V., Spirakis, P.: $\mathcal{NP}$-Completeness Results and Efficient Approximations for Radiocoloring in Planar Graphs. In: Proceedings of the 25th International Symposium on Mathematical Foundations of Computer Science, Lecture Notes of Computer Science, vol. 1893, pp. 363–372. Springer (2000)
6. Fotakis, D., Nikoletseas, S., Papadopoulou, V.G., Spirakis, P.G.: Radiocoloring in Planar Graphs: Complexity and Approximations. Theor. Comput. Sci. Elsevier **340**, 514–538 (2005)
7. Fotakis, D., Pantziou, G., Pentaris, G., Spirakis, P.: Frequency Assignment in Mobile and Radio Networks. In: Networks in Distributed Computing, DIMACS Series in Discrete Mathematics and Theoretical Computer Science 45, pp. 73–90 (1999)
8. Griggs, J., Liu, D.: Minimum Span Channel Assignments. In: Recent Advances in Radio Channel Assignments, Invited Minisymposium, Discrete Mathematics (1998)
9. van d. Heuvel, J., McGuiness, S.: Colouring the Square of a Planar Graph. CDAM Research Report Series, July 1999
10. Jerrum, M.: A very simple Algorithm for Estimating the Number of k-colourings of a Low Degree Graph. Random Struct. Algorithms **7**, 157–165 (1994)
11. Lin, Y.L., Skiena, S.: Algorithms for Square Roots of Graphs. SIAM J. Discret. Math. **8**, 99–118 (1995)
12. Ramanathan, S., Loyd, E.R.: The Complexity of Distance 2-Coloring. In: Proceedings of the 4th International Conference of Computing and Information, pp. 71–74 (1992)

Randomization in Distributed Computing

1996; Chandra

TUSHAR DEEPAK CHANDRA
IBM Watson Research Center, Yorktown Heights, NY, USA

Keywords and Synonyms

Agreement; Byzantine agreement

Problem Definition

This problem is concerned with using the multi-writer multi-reader register primitive in the shared memory model to design a fast, wait-free implementation of consensus. Below are detailed descriptions of each of these terms.

Consensus Problems

There are n processors and the goal is to design distributed algorithms to solve the following two consensus problems for these processors.

Problem 1 (Binary consensus)
Input: Processor i has input bit b_i.
Output: *Each processor i has output bit b'_i such that: 1) all the output bits b'_i equal the same value v; and 2) $v = b_i$ for some processor i.*

Problem 2 (Id consensus)
Input: *Processor i has a unique id u_i.*
Output: *Each processor i has output value u'_i such that: 1) all the output values u'_i equal the same value u; and 2) $u = u_i$ for some processor i.*

Wait-Free

This result builds on extensive previous work on the shared memory model of parallel computing. Shared object types include data structures such as read/write registers and synchronization primitives such as “test and set”.

A shared object is said to be *wait-free* if it ensures that every invocation on the object is guaranteed a response in finite time even if some or all of the other processors in the system crash. In this problem, the existence of wait-free registers is assumed and the goal is to create a fast wait-free algorithm to solve the consensus problem. In the rest of this summary, "wait-free implementations" will be referred to simply as "implementations" i. e. the term wait-free will be omitted.

Multi-writer Multi-reader Register

Many past results on solving consensus in the shared memory model assume the existence of a single writer multi-reader register. For such a register, there is a single writer client and multiple reader clients. Unfortunately, it is easy to show that the per processor step complexity of any implementation of consensus from single writer multi-reader registers will be at least linear in the number of processors. Thus, to achieve a time efficient implementation of consensus, the more powerful primitive of a multi-writer multi-reader register must be assumed. A multi-writer multi-reader register assumes the clients of the register are multiple writers and multiple readers. It is well known that it is possible to implement such a register in the shared memory model.

The Adversary

Solving the above problems is complicated by the fact that the programmer has little control over the rate at which individual processors execute. To model this fact, it is assumed that the schedule at which processors run is picked by an adversary. It is well-known that there is no deterministic algorithm that can solve either Binary consensus or ID consensus in this adversarial model if the number of processors is greater than 1 [6,7]. Thus, researchers have turned to the use of randomized algorithms to solve this problem [1]. These algorithms have access to random coin flips. Three types of adversaries are considered for randomized algorithms. The *strong adversary* is assumed to know the outcome of a coin flip immediately after the coin is flipped and to be able to modify its schedule accordingly. The *oblivious adversary* has to fix the schedule before any of the coins are flipped. The *intermediate adversary* is not permitted to see the outcome of a coin flip until some process makes a choice based on that coin flip. In particular, a process can flip a coin and write the result in a global register, but the intermediate adversary does not know the outcome of the coin flip until some process reads the value written in the register.

Key Results

Theorem 1 *Assuming the existence of multi-writer multi-reader registers, there exists a randomized algorithm to solve binary consensus against an intermediate adversary with* $O(1)$ *expected steps per processor.*

Theorem 2 *Assuming the existence of multi-writer multi-reader registers, there exists a randomized algorithm to solve id-consensus against an intermediate adversary with* $O(\log^2 n)$ *expected steps per processor.*

Both of these results assume that every processor has a unique identifier. Prior to this result, the fastest known randomized algorithm for binary consensus made use of single writer multiple reader registers, was robust against a strong adversary, and required $O(n \log^2 n)$ steps per processor [2]. Thus, the above improvements are obtained at the cost of weakening the adversary and strengthening the system model when compared to [2].

Applications

Binary consensus is one of the most fundamental problems in distributed computing. An example of its importance is the following result shown by Herlihy [8]: If an abstract data type X together with shared memory is powerful enough to implement wait-free consensus, then X together with shared memory is powerful enough to implement in a wait-free manner any other data structure Y. Thus, using this result, a wait-free version of any data structure can be created using only wait-free multi-writer multi-reader registers as a building block.

Binary consensus has practical applications in many areas including: database management, multiprocessor computation, fault diagnosis, and mission-critical systems such as flight control. Lynch contains an extensive discussion of some of these application areas [9].

Open Problems

This result leaves open several problems. First, it leaves open a gap on the number of steps per process required to perform randomized consensus using multi-writer multi-reader registers against the *strong* adversary. A recent result by Attiya and Censor shows an $\Omega(n^2)$ lower bound on the total number of steps for all processors with multi-writer multi-reader registers (implying $\Omega(n)$ steps per process) [3]. They also show a matching upper bound of $O(n^2)$ on the total number of steps. However, closing the gap on the per-process number of steps is still open.

Another open problem is whether there is a randomized implementation of id consensus using multi-reader multi-writer registers that is robust to the intermediate adversary and whose expected number of steps per processor is better than $O(\log^2 n)$. In particular, is a constant run time possible? Aumann in follow up work to this result was able to improve the expected run time per process to $O(\log n)$ [4]. However, to the best of the reviewer's knowledge, there have been no further improvements.

A third open problem is to close the gap on the time required to solve binary consensus against the strong adversary with a single writer multiple reader register. The fastest known randomized algorithm in this scenario requires $O(n \log^2 n)$ steps per processor [2]. A trivial lower bound on the number of steps per processor when single-writer registers are used is $\Omega(n)$. However, to the best of this reviewers knowledge, a $O(\log^2 n)$ gap still remains open.

A final open problem is to close the gap on the total work required to solve consensus with single-reader single-writer registers against an oblivious adversary. Aumann and Kapah-Levy describe algorithms for this scenario that require $O(n \log n \exp(2\sqrt{\ln n \ln(c \log n \log^* n)}$ expected total work for some constant c [5]. In particular, the total work is less than $O(n^{1+\epsilon})$ for any $\epsilon > 0$. A trivial lower bound on total work is $\Omega(n)$, but a gap remains open.

Cross References

- ▶ Asynchronous Consensus Impossibility
- ▶ Atomic Broadcast
- ▶ Byzantine Agreement
- ▶ Implementing Shared Registers in Asynchronous Message-Passing Systems
- ▶ Optimal Probabilistic Synchronous Byzantine Agreement
- ▶ Registers
- ▶ Set Agreement
- ▶ Snapshots in Shared Memory
- ▶ Wait-Free Synchronization

Recommended Reading

1. Aspnes, J.: Randomized protocols for asynchronous consensus. Distrib. Comput. **16**(2–3), 165–175 (2003)
2. Aspnes, J., Waarts, O.: Randomized consensus in expected $o(n \log^2 n)$ operations per processor. In: Proceedings of the 33rd Symposium on Foundations of Computer Science. 24–26 October 1992, pp. 137–146. IEEE Computer Society, Pittsburgh (1992)
3. Attiya, H., Censor, K.: Tight bounds for asynchronous randomized consensus. In: Proceedings of the Symposium on the Theory of Computation. San Diego, 11–13 June 2007 ACM Special Interest Group on Algorithms and Computation Theory (SIGACT) (2007)
4. Aumann, Y.: Efficient asynchronous consensus with the weak adversary scheduler. In: Symposium on Principles of Distrib. Comput.(PODC) Santa Barbara, 21–24 August 1997, pp. 209–218. ACM Special Interest Group on Algorithms and Computation Theory (SIGACT) (1997)
5. Aumann, Y., Kapach-Levy, A.: Cooperative sharing and asynchronous consensus using single-reader/single-writer registers. In: Proceedings of 10th Annual ACM-SIAM Symposium of Discrete Algorithms (SODA) Baltimore, 17–19 January 1999, pp. 61–70. Society for Industrial and Applied Mathematics (SIAM) (1999)
6. Dolev, D., Dwork, C., Stockmeyer, L.: On the minimal synchronism needed for distributed consensus. J. ACM (JACM) **34**(1), 77–97 (1987)
7. Fischer, M.J., Lynch, N.A., Paterson, M.: Impossibility of distributed consensus with one faulty process. In: Proceedings of the 2nd ACM SIGACT-SIGMOD Symposium on Principles of Database System (PODS) Atlante, 21–23 March, pp. 1–7. Association for Computational Machinery (ACM) (1983)
8. Herlihy, M.: Wait-free synchronization. ACM Trans. Programm. Lang. Syst. **13**(1), 124–149 (1991)
9. Lynch, N.: Distributed Algorithms. Morgan Kaufmann, San Mateo (1996)

Randomized Broadcasting in Radio Networks

1992; Reuven Bar-Yehuda, Goldreich, Itai

ALON ITAI
Depart of Computer Science, Technion,
Haifa, Israel

Keywords and Synonyms

Multi-hop radio networks; Ad hoc networks

Problem Definition

The paper investigates deterministic and randomized protocols for achieving broadcast (distributing a message from a source to all other nodes) in arbitrary multi-hop synchronous radio networks.

The model consists of an arbitrary (undirected) network, with processors communicating in synchronous time-slots subject to the following rules. In each time-slot, each processor acts either as a *transmitter* or as a *receiver*. A processor acting as a receiver is said to receive a message in time-slot t if exactly one of its neighbors transmits in that time-slot. The message received is the one transmitted. If more than one neighbor transmits in that time-slot, a *conflict* occurs. In this case the receiver may either get a message from one of the transmitting neighbors or get no message. It is assumed that conflicts (or "collisions") are not detected, hence a processor cannot distinguish the case in which no neighbor transmits from the case in which two or more of its neighbors transmits during that time-slot. The processors are not required to have ID's nor do they know their neighbors, in particular the processors do not know the topology of the network.

The only inputs required by the protocol are the number of processors in the network – n, Δ – an a priori known upper bound on the maximum degree in the network and the error bound –ϵ. (All bounds are a priori known to the algorithm.)

Broadcast is a task initiated by a single processor, called the *source*, transmitting a single *message*. The goal is to have the message reach all processors in the network.

Key Results

The main result is a randomized protocol that achieves broadcast in time which is optimal up to a logarithmic factor. In particular, with probability $1 - \epsilon$, the protocol achieves broadcast within $O((D + \log n/\epsilon) \cdot \log n)$ time-slots.

On the other hand, a linear lower bound on the deterministic time-complexity of broadcast is proved. Namely, any deterministic broadcast protocol requires $\Omega(n)$ time-slots, even if the network has diameter 3, and n is known to all processors. These two results demonstrate an exponential gap in complexity between randomization and determinism.

Randomized Protocols

The Procedure *Decay* The basic idea used in the protocol is to resolve potential conflicts by randomly eliminating half of the transmitters. This process of "cutting by half" is repeated each time-slot with the hope that there will exist a time-slot with a single active transmitter. The "cutting by half" process is easily implemented distributively by letting each processor decide randomly whether to eliminate itself. It will be shown that if all neighbors of a receiver follow the elimination procedure then with positive probability there exists a time slot in which exactly one neighbor transmits.

What follows is a description of the procedure for sending a message m, that is executed by each processor after receiving m:

```
procedure Decay(k, m);
  repeat at most k times (but at least once!)
    send m to all neighbors;
    set coin ← 0 or 1 with equal probability.
  until coin = 0.
```

By using elementary probabilistic arguments, one can prove:

Theorem 1 *Let y be a vertex of G. Also let $d \geq 2$ neighbors of y execute Decay during the time interval $[0, k)$ and assume that they all start the execution at $Time = 0$. Then $P(k, d)$, the probability that y receives a message by $Time = k$, satisfies:*
1. $\lim_{k\to\infty} P(k, d) \geq \frac{2}{3}$;
2. for $k \geq 2\lceil \log d \rceil$, $P(k, d) > \frac{1}{2}$.
(All logarithms are to base 2.)

The expected termination time of the algorithm depends on the probability that $coin = 0$. Here, this probability is set to be one half. An analysis of the merits of using other probabilities was carried out by Hofri [4].

The Broadcast Protocol The broadcast protocol makes several calls to $Decay(k, m)$. By Theorem 1 (2), to ensure that the probability of a processor y receiving the message be at least 1/2, the parameter k should be at least $2 \log d$ (where d is the number of neighbors sending a message to y). Since d is not known, the parameter was chosen as $k = 2\lceil \log \Delta \rceil$ (recall that Δ was defined to be an upper bound on the in-degree). Theorem 1 also requires that all participants start executing *Decay* at the same time-slot. Therefore, *Decay* is initiated only at integer multiples of $2\lceil \log \Delta \rceil$.

```
procedure Broadcast;
  $k = 2\lceil \log \Delta \rceil$;
  $t = 2\lceil \log(N/\epsilon) \rceil$;
  Wait until receiving a message, say $m$;
  do $t$ times {
    Wait until ($Time$ mod $k$) = 0 ;
    Decay($k$, $m$) ;
  }
```

A network is said to execute the *Broadcast_scheme* if some processor, denoted s, transmits an initial message and each processor executes the above *Broadcast* procedure.

Theorem 2 *Let $T = 2D + 5 \max\{\sqrt{D}, \sqrt{\log(n/\epsilon)}\} \cdot \sqrt{\log(n/\epsilon)}$. Assume that Broadcast_scheme starts at $Time = 0$. Then, with probability $\geq 1 - 2\epsilon$, by time $2\lceil \log \Delta \rceil \cdot T$ all nodes will receive the message. Furthermore, with probability $\geq 1 - 2\epsilon$, all the nodes will terminate by time $2\lceil \log \Delta \rceil \cdot (T + \lceil \log(N/\epsilon) \rceil)$.*

The bound provided by Theorem 2 contains two additive terms: the first represents the diameter of the network and the second represents delays caused by conflicts (which are rare, yet they exist).

Additional Properties of the Broadcast Protocol:

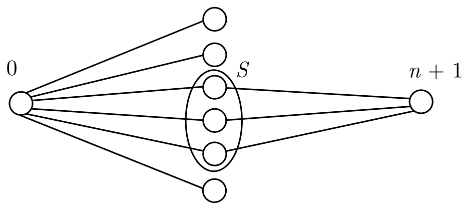
- **Processor IDs** – The protocol does not use processor IDs, and thus does not require that the processors have distinct IDs (or that they know the identity of their neighbors). Furthermore, a processor is not even required to know the number of its neighbors. This property makes the protocol adaptive to changes in topology which occur throughout the execution, and resilient to non-malicious faults.

- **Knowing the size of the network** – The protocol performs almost as well when given instead of the actual number of processors (i. e., n), a "good" upper bound on this number (denoted N). An upper bound polynomial in n yields the same time-complexity, up to a constant factor (since complexity is logarithmic in N).
- **Conflict detection** – The algorithm and its complexity remain valid even if no messages can be received when a conflict occurs.
- **Simplicity and fast local computation** – In each time slot each processor performs a constant amount of local computation.
- **Message complexity** – Each processor is active for $\lceil \log(N/\epsilon) \rceil$ consecutive phases and the average number of transmissions per phase is at most 2. Thus the expected number of transmissions of the entire network is bounded by $2n \cdot \lceil \log(N/\epsilon) \rceil$.
- **Adaptiveness to changing topology and fault resilience** – The protocol is resilient to some changes in the topology of the network. For example, edges may be added or deleted at any time, provided that the network of unchanged edges remains connected. This corresponds to fail/stop failure of edges, thus demonstrating the resilience to some non-malicious failures.
- **Directed networks** – The protocol does not use acknowledgments. Thus it may be applied even when the communication links are not symmetric, i. e., the fact that processor v can transmit to u does not imply that u can transmit to v. (The appropriate network model is, therefore, a directed graph.) In real life this situation occurs, for instance, when v has a stronger transmitter than u.

A Lower Bound on Deterministic Algorithms

For deterministic algorithms one can show a lower bound: for every n there exist a family of n-node networks such that every deterministic broadcast scheme requires $\Omega(n)$ time. For every **non-empty** subset $S \subseteq \{1, 2, \dots, n\}$, consider the following network G_S (Fig. 1).

Node 0 is the *source* and node $n+1$ the *sink*. The source initiates the message and the problem of broadcast in G_S is to reach the sink. The difficulty stems from the fact that the partition of the middle layer (i. e., S) is not known a priori. The following theorem can be proved by a series



Randomized Broadcasting in Radio Networks, Figure 1
The network used for the lower bound

of reductions to a certain "hitting game":

Theorem 3 *Every deterministic broadcast protocol that is correct for all n-node networks requires time $\Omega(n)$.*

In [2] there was some confusion concerning the broadcast model. In that paper it was erroneously claimed that the lower bound holds also when a collision is indistinguishable from the absence of transmission. Kowalski and Pelc [5] disproved this claim by showing how to broadcast in logarithmic time on all networks of type G_S.

Applications

The procedure *Decay* has been used to resolve contention in radio and cellular phone networks.

Cross Reference

▶ Broadcasting in Geometric Radio Networks
▶ Communication in Ad Hoc Mobile Networks Using Random Walks
▶ Deterministic Broadcasting in Radio Networks
▶ Randomized Gossiping in Radio Networks

Recommended Reading

Subsequent papers showed the optimality of the randomized algorithm:

- Alon et al. [1] showed the existence of a family of radius-2 networks on n vertices for which any broadcast schedule requires at least $\Omega(\log^2 n)$ time slots.
- Kushilevitz and Mansour [7] showed that for any randomized broadcast protocol there exists a network in which the expected time to broadcast a message is $\Omega(D \log(N/D))$.
- Bruschi and Del Pinto [3] showed that for any deterministic distributed broadcast algorithm, any n and $D \leq n/2$ there exists a network with n nodes and diameter D such that the time needed for broadcast is $\Omega(D \log n)$.
- Kowalski and Pelc [6] discussed networks in which collisions are indistinguishable from the absence of trans-

mission. They showed an $\Omega(n \log n / \log(n/D))$ lower bound and an $O(n \log n)$ upper bound. For this model, they also showed an $O(D \log n + \log^2 n)$ randomized algorithm, thus matching the lower bound of [1] and improving the bound of [2] for graphs for which $D = \theta(n/\log n)$.

1. Alon, N., Bar-Noy, A., Linial, N., Peleg, D.: A lower bound for radio broadcast. J. Comput. Syst. Sci. **43**(2), 290–298 (1991)
2. Bar-Yehuda, R., Goldreich, O., Itai, A.: On the time-complexity of broadcast in multi-hop radio networks: An exponential gap between determinism and randomization. J. Comput. Syst. Sci. **45**(1), 104–126 (1992)
3. Bruschi, D., Del Pinto, M.: Lower bounds for the broadcast problem in mobile radio networks. Distrib. Comput. **10**(3), 129–135 (1997)
4. Hofri, M.: A feedback-less distributed broadcast algorithm for multihop radio networks with time-varying structure. In: Computer Performance and Reliability, pp. 353–368. (1987)
5. Kowalski, D.R., Pelc, A.: Deterministic broadcasting time in radio networks of unknown topology. In: FOCS '02: Proceedings of the 43rd Symposium on Foundations of Computer Science, Washington, DC, USA, pp. 63–72. IEEE Computer Society (2002)
6. Kowalski, D.R., Pelc, A.: Broadcasting in undirected ad hoc radio networks. Distrib. Comput. **18**(1), 43–57 (2005)
7. Kushilevitz, E., Mansour, Y.: An $\Omega(d \log(n/d))$ lower bound for broadcast in radio networks. In: PODC, 1993, pp. 65–74

Randomized Energy Balance Algorithms in Sensor Networks

2005; Leone, Nikoletseas, Rolim

Pierre Leone[1], Sotiris Nikoletseas[2], José Rolim[1]
[1] Informatics Department, University of Geneva, Geneva, Switzerland
[2] Computer Engineering and Informatics, Department and CTI, University of Patras, Patras, Greece

Keywords and Synonyms

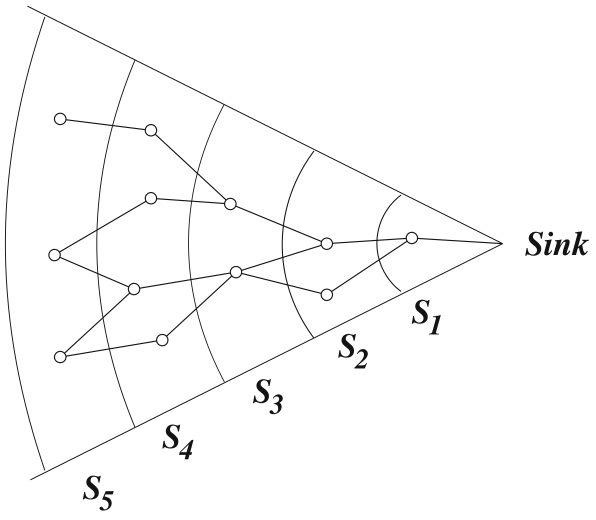
Power conservation

Problem Definition

Recent developments in wireless communications and digital electronics have led to the development of extremely small in size, low-power, low-cost sensor devices (often called smart dust). Such tiny devices integrate sensing, data processing and wireless communication capabilities. Examining each such resource constraint device individually might appear to have small utility; however, the distributed self-collaboration of large numbers of such devices into an ad hoc network may lead to the efficient accomplishment of large sensing tasks i. e., reporting data about the realization of a local event happening in the network area to a faraway control center.

The problem considered is the development of a randomized algorithm to balance energy among sensors whose aim is to detect events in the network area and report them to a sink. The network is sliced by the algorithm into layers composed of sensors at approximately equal distances from the sink [1,2,8] (Fig. 1). The slicing of the network depends on the communication distance. The sink initiates the process by sending a control message containing a counter, the value of which is initially 1. Sensors receiving the message assign themselves to a slice number corresponding to the counter, increment the counter and propagate the message in the network. A sensor already assigned to a slice ignores subsequent received control messages.

The strategy suggested to balance the energy among sensors consists in allowing a sensor to probabilistically choose between either sending data to a sensor in the next layer towards the sink or sending the data directly to the sink. The difference between the two choices is the energy consumption, which is much higher if the sensor decides to report to the sink directly. The energy consumption is modeled as a function of the transmission distance by assuming that the energy necessary to send data up to a distance d is proportional to d^2. Actually, more accurate models can be considered, in which the dependence is of the form d^α, with $2 \leq \alpha \leq 5$ depending on the particular environmental conditions. Although the model chosen



Randomized Energy Balance Algorithms in Sensor Networks, Figure 1
The sink and five slices $S_1, \ldots, S_5$

determines the parameters of the algorithm, the particular shape of the function describing the relationship between the distance of transmission and energy consumption is not relevant except that it might increase with distance. The distance between two successive slices is normalized to be 1. Hence, a sensor sending data to one of its neighbors consumes one unit of energy and a sensor located in slice i consumes i^2 units of energy to report to the sink directly. Small hop transmissions are cheap (with respect to energy consumption) but pass through the critical region around the sink and might strain sensors in that region, while expensive direct transmissions bypass that critical area.

Energy balance is defined as follows:

Definition 1 The network is energy-balanced if the average per sensor energy dissipation is the same for all sectors, i. e., when

$$\frac{E[\mathcal{E}_i]}{S_i} = \frac{E[\mathcal{E}_j]}{S_j}\,, \qquad i, j = 1, \ldots, n \tag{1}$$

where $\mathcal{E}_i$ is the total energy available and S_i is the number of nodes in slice number i.

The dynamics of the network is modeled by assigning probabilities λ_i, $i = 1, \ldots, N$, $\sum \lambda_i = 1$, of the occurrence of an event in slice i. The protocol consists in transmitting the data to a neighbor slice with probability p_i and with probability $1 - p_i$ to the sink, for a sensor belonging to slice i. Hence, the mean energy consumption per data unit is $p_i + (1 - p_i)i^2$. A central assumption in the following is that the events are evenly generated in a given slice. Then, denoting by e_i the energy available per node in slice i (i. e., $e_i = \mathcal{E}_i/S_i$), the problem of energy-balanced data propagation can be formally stated as follows:

Given λ_i, e_i, S_i, $i = 1, \ldots, N$, find p_i, λ such that

$$\begin{aligned}&\underbrace{\left(\lambda_i + \lambda_{i+1}p_{i+1} + \ldots + \lambda_n p_n p_{n-1} \cdots p_{i+1}\right)}_{=:x_i}\\ &\cdot \left(p_i \frac{1}{S_i} + (1 - p_i)\frac{i^2}{S_i}\right) = \lambda e_i\,, \quad i = 1, \ldots, N\,.\end{aligned} \tag{2}$$

Equation (2) amounts to ensuring that the mean energy dissipation for all sensors is proportional to the available energy. In turn, this ensures that sensors might, on average, run out of energy all at the same time. Notice that (2) contains the definitions of the x_i. They are the ones estimated in the pseudo-code in Fig. 2, the successive estimations being denoted as $\tilde{x}_i$. These variables are proportional to the number of messages handled by slice i.

```
Initialize $\tilde{x}_0 = \lambda, \ldots, \tilde{x}_n$
Initialize NbrLoop=1
**repeat** forever
  Send $\tilde{x}_i$ and $\lambda$ values to the stations which compute
                                              their $p_i$ probability
  wait for a data
  **for** i=0 **to** n
    **if** the data passed through slice $i$ **then**
      $X \leftarrow 1$
    **else**
      $X \leftarrow 0$
    **end if**
    Generate $R$ a $\tilde{x}_i$-Bernoulli random variable
    $\tilde{x}_i \leftarrow \tilde{x}_i + \frac{1}{NbrLoop}(X - R)$
    Increment $NbrLoop$ by one.
  **end for**
**end repeat**
```

Randomized Energy Balance Algorithms in Sensor Networks, Figure 2
Pseudo-code for estimation of the x_i value by the sink

Key Results

In [1,2] recursive equations similar to (2) were suggested and solved in closed form under adequate hypotheses. The need for a priori knowledge of the probability of occurrence of the events, the λ_i parameters, was considered in [7], in which these parameters were estimated by the sink on the basis of the observations of the various paths the data follow. The algorithm suggested is based on recursive estimation, is computationally not expensive and converges with rate $\mathcal{O}(1/\sqrt{n})$. One might argue that the rate of convergence is slow; however, it is numerically observed that relatively quickly compared with the convergence time, the algorithm finds an estimation close enough to the final value. The estimation algorithm run by the sink (which has no energy constraints) is given in Fig. 2.

Results taken from [1,2,7] all assume the existence of an energy-balance solution. However, particular distributions of the events might prevent the existence of such a solution and the relevant question is no longer the computation of an energy-balance algorithm. For instance, assuming that $\lambda_N = 0$, sensors in slice N have no way of balancing energy. In [9] the problem was reformulated as finding the probability distribution $\{p_i\}_{i=1,\ldots,N}$ which leads to the maximal functional lifetime of the networks. It was proved that if an energy-balance strategy exists, then it maximizes the lifetime of the network establishing formally the intuitive reasoning which was the motivation to consider energy-balance strategies. A centralized algorithm was presented to compute the optimal parameters. Moreover, it was observed numerically that the interslice energy consumption is prone to be uneven and a spreading technique was suggested and numerically validated as being efficient to overcome this limitation of the probabilistic algorithm.

The communication graph considered is a restrictive subset of the complete communication graph and it is legitimate to wonder whether one can improve the situation by extending it. For instance, by allowing data to be sent two hops or more away. In [3,6] it was proved that the topology in which sensors communicate only to neighbor slices and the sink is the one which maximizes the flow of data in the network. Moreover, the communication graph in which sensors send data only to their neighbors and the sink leads to a completely distributed algorithm balancing energy [6]. Indeed, as a sensor sends data to a neighbor slice, the neighbor must in turn send the data and can attach information concerning its own energy level. This information might be captured by the initial sensor since it belongs to the communication range of its neighbor (this does not hold any longer if multiple hops are allowed). Hence, a distributed strategy consists in sending data to a particular neighbor only if its energy level consumption is lower, otherwise the data are sent directly to the sink.

Applications

Among the several constraints sensor networks designers have to face, energy management is central since sensors are usually battery powered, making the lifetime of the networks highly sensitive to the energy management. Besides the traditional strategy consisting in minimizing the energy consumption at sensor nodes, energy-balance schemes aim at balancing the energy consumption among sensors. The intuitive function of such schemes is to avoid energy depletion holes appearing as some sensors that run out of their available energy resources and are no longer able to participate in the global function of the networks. For instance, routing might be no longer possible if a small number of sensors run out of energy, leading to a disconnected network. This was pointed out in [5] as well as the need to develop application-specific protocols. Energy balancing is suggested as a solution in order to make the global functional lifetime of the network longer. The earliest development of dedicated protocols ensuring energy balance can be found in [4,10,11].

A key application is to maximize the lifetime of the network while gathering data to a sink. Besides increasing the lifetime of the networks, other criteria have to be taken into account. Indeed, the distributed algorithm might be as simple as possible owing to limited computational resources, might avoid collisions or limit the total number of transmissions, and might ensure a large enough flow of data from the sensors toward the sink. Actually, maximizing the flow of data is equivalent to maximizing the lifetime of sensor networks if some particular realizable conditions are fulfilled. Besides the simplicity of the distributed algorithm, the network deployment and the self-realization of the network structure might be possible in realistic conditions.

Cross References

► Obstacle Avoidance Algorithms in Wireless Sensor Networks
► Probabilistic Data Forwarding in Wireless Sensor Networks

Recommended Reading

1. Efthymiou, C., Nikoletseas, S., Rolim, J.: Energy Balanced Data Propagation in Wireless Sensor Networks. 4th International Workshop on Algorithms for Wireless, Mobile, Ad-Hoc and Sensor Networks (WMAN '04) IPDPS 2004, Wirel. Netw. J. (WINET) **12**(6), 691–707 (2006)
2. Efthymiou, C., Nikoletseas, S., Rolim, J.: Energy Balanced Data Propagation in Wireless Sensor Networks. In: Wireless Networks (WINET) Journal, Special Issue on Algorithms for Wireless, Mobile, Ad Hoc and Sensor Networks. Springer (2006)
3. Giridhar, A., Kumar, P.R.: Maximizing the Functional Lifetime of Sensor Networks. In: Proceedings of The Fourth International Conference on Information Processing in Sensor Networks, IPSN '05, UCLA, Los Angeles, April 25–27 2005
4. Guo, W., Liu, Z., Wu, G.: An Energy-Balanced Transmission Scheme for Sensor Networks. In: 1st ACM International Conference on Embedded Networked Sensor Systems (ACM SenSys 2003), Poster Session, Los Angeles, CA, November 2003
5. Heinzelman, W., Chandrakasan, A., Balakrishnan, H.: Energy-efficient communication protocol for wireless microsensor networks. In: Proceedings of the 33rd IEEE Hawaii International Conference on System Sciences (HICSS 2000). 2000
6. Jarry, A., Leone, P., Powell, O., Rolim, J.: An Optimal Data Propagation Algorithm for Maximizing the Lifespan of Sensor Networks. In: Second International Conference, DCOSS 2006, San Francisco, CA, USA, June 2006. Lecture Notes in Computer Science, vol. 4026, pp. 405–421. Springer, Berlin (2006)
7. Leone, P., Nikoletseas, S., Rolim, J.: An Adaptive Blind Algorithm for Energy Balanced Data Propagation in Wireless Sensor Networks. In: First International Conference on Distributed Computing in Sensor Systems (DCOSS), Marina del Rey, CA, USA, June/July 2005. Lecture Notes in Computer Science, vol. 3560, pp. 35–48. Springer, Berlin (2005)
8. Olariu, S., Stojmenovic, I.: Design guidelines for maximizing lifetime and avoiding energy holes in sensor networks with uniform distribution and uniform reporting. In: IEEE INFOCOM, Barcelona, Spain, April 24–25 2006
9. Powell, O., Leone, P., Rolim, J.: Energy Optimal Data Propagation in Sensor Networks. J. Prarallel Distrib. Comput. **67**(3), 302–317 (2007) http://arxiv.org/abs/cs/0508052

10. Singh, M., Prasanna, V.: Energy-Optimal and Energy-Balanced Sorting in a Single-Hop Wireless Sensor Network. In: Proc. First IEEE International Conference on Pervasive Computing and Communications (PerCom '03), pp. 302–317, Fort Worth, 23–26 March 2003
11. Yu, Y., Prasanna, V.K.: Energy-Balanced Task Allocation for Collaborative Processing in Networked Embedded System. In: Proceedings of the 2003 Conference on Language, Compilers, and Tools for Embedded Systems (LCTES'03), pp. 265–274, San Diego, 11–13 June 2003

Randomized Gossiping in Radio Networks

2001; Chrobak, Gąsieniec, Rytter

Leszek Gąsieniec
Department of Computer Science,
University of Liverpool, Liverpool, UK

Keywords and Synonyms

Wireless networks; Broadcast; Gossip; Total exchange of information; All-to-all communication

Problem Definition

The two classical problems of disseminating information in computer networks are *broadcasting* and *gossiping*. In broadcasting, the goal is to distribute a message from a distinguished *source* node to all other nodes in the networks. In gossiping, each node v in the network initially contains a message m_v, and the task is to distribute each message m_v to all nodes in the network.

The radio network abstraction captures the features of distributed communication networks with multi-access channels, with minimal assumptions on the channel model and processors' knowledge. Directed edges model uni-directional links, including situations in which one of two adjacent transmitters is more powerful than the other. In particular, there is no feedback mechanism (see, for example, [6]). In some applications, collisions may be difficult to distinguish from the noise that is normally present in the channel, justifying the need for protocols that do not depend on the reliability of the collision detection mechanism (see [3,4]). Some network configurations are subject to frequent changes. In other networks, a network topology could be unstable or dynamic; for example, when mobile users are present. In such situations, algorithms that do not assume any specific topology are more desirable.

More formally a radio network is a directed graph $G = (V, E)$, where by $|V| = n$ we denote the number of nodes in this graph. Individual nodes in V are denoted by letters $u, v, \ldots$. If there is an edge from u to v, i. e., $(u, v) \in E$, then we say that v is an *out-neighbor* of u and u is an *in-neighbor* of v. Messages are denoted by letter m, possibly with indices. In particular, the message originating from node v is denoted by m_v. The whole set of initial messages is $M = \{m_v : v \in V\}$. During the computation, each node v holds a set of messages M_v that have been received by v so far. Initially, each node v does not possess any information apart from $M_v = \{m_v\}$. Without loss of generality, whenever a node is in the transmitting mode, one can assume that it transmits the whole content of M_v.

The time is divided into discrete time steps. All nodes start simultaneously, have access to a common clock, and work synchronously. A gossiping algorithm is a protocol that for each node u, given all past messages received by u, specifies, for each time step t, whether u will transmit a message at time t, and if so, it also specifies the message. A message M transmitted at time t from a node u is sent instantly to all its out-neighbors. An out-neighbor v of u receives M at time step t only if no collision occurred, that is, if the other in-neighbors of v do not transmit at time t at all. Further, collisions cannot be distinguished from background noise. If v does not receive any message at time t, it knows that either none of its in-neighbors transmitted at time t, or that at least two did, but it does not know which of these two events occurred. The *running time* of a gossiping algorithm is the smallest t such that for any network topology, and any assignment of identifiers to the nodes, all nodes receive messages originating in every other node no later than at step t.

Limited Broadcast$_v$(k) Given an integer k and a node v, the goal of *limited broadcasting* is to deliver the message m_v (originating in v) to at least k other nodes in the network.

Distributed Coupon Collection The set of network nodes V can be interpreted as a set of n bins and the set of messages M as a set of n coupons. Each coupon has at least k copies, each copy belonging to a different bin. M_v is the set of coupons in bin v. Consider the following process. At each step, one opens every bin at random, independently, with probability $1/n$. If no bin is opened, or if two or more bins are opened, a failure occurs and no coupons are collected. If exactly one bin, say v, is opened, all coupons from M_v are collected. The task is to establish how many steps are needed to collect (a copy of) each coupon.

Key Results

Theorem 1 ([1]) *There exists a deterministic $O(k \log^2 n)$-time algorithm for limited broadcasting from any node in radio networks with an arbitrary topology.*

Theorem 2 ([1]) *Let δ be a given constant, $0 < \delta < 1$, and $s = (4n/k)\ln(n/\delta)$. After s steps of the distributed coupon collection process, with probability at least $1-\delta$, all coupons will be collected.*

Theorem 3 ([1]) *Let ϵ be a given constant, where $0 < \epsilon < 1$. There exists a randomized $O(n\log^3 n\log(n/\epsilon))$-time Monte Carlo-type algorithm that completes radio gossiping with probability at least $1-\epsilon$.*

Theorem 4 ([1]) *There exists a randomized Las Vegas-type algorithm that completes radio gossiping with expected running time $O(n\log^4 n)$.*

Applications

Further work on efficient randomized radio gossiping include the $O(n\log^3 n)$-time algorithm by Liu and Prabhakaran, see [5], where the deterministic procedure for limited broadcasting is replaced by its $O(k\log n)$-time randomized counterpart. This bound was later reduced to $O(n\log^2 n)$ by Czumaj and Rytter in [2], where a new randomized limited broadcasting procedure with an expected running time $O(k)$ is proposed.

Open Problems

The exact complexity of randomized radio gossiping remains an open problem. All three gossiping algorithms [1,2,5] are based on the concepts of limited broadcast and distributed coupon collection. The two improvements [2,5] refer solely to limited broadcasting. Thus, very likely further reduction of the time complexity must coincide with more accurate analysis of the distributed coupon collection process or with development of a new gossiping procedure.

Recommended Reading

1. Chrobak, M., Gąsieniec, L., Rytter, W.: A Randomized Algorithm for Gossiping in Radio Networks. In: Proc. 8th Annual International Computing Combinatorics Conference. Guilin, China, pp. 483–492 (2001) Full version in Networks **43**(2), 119–124 (2004)
2. Czumaj, A., Rytter, W.: Broadcasting algorithms in radio networks with unknown topology. J. Algorithms **60**(2), 115–143 (2006)
3. Ephremides, A., Hajek, B.: Information theory and communication networks: an unconsummated union. IEEE Trans. Inf. Theor. **44**, 2416–2434 (1998)
4. Gallager, R.: A perspective on multiaccess communications. IEEE Trans. Inf. Theor. **31**, 124–142 (1985)
5. Liu, D., Prabhakaran, M.: On randomized broadcasting and gossiping in radio networks. In: Proc. 8th Annual International Computing Combinatorics Conference, pp. 340-349, Singapore (2002)
6. Massey, J.L., Mathys, P.: The collision channel without feedback. IEEE Trans. Inf. Theor. **31**, 192–204 (1985)

Randomized Minimum Spanning Tree

1995; Karger, Klein, Tarjan

VIJAYA RAMACHANDRAN
Department of Computer Science,
University of Texas at Austin, Austin, TX, USA

Problem Definition

The input to the problem is a connected undirected graph $G = (V, E)$ with a weight $w(e)$ on each edge $e \in E$. The goal is to find a spanning tree of minimum weight, where for any subset of edges $E' \subseteq E$, the *weight of* E' is defined to be $w(E') = \sum_{e\in E'} w(e)$.

If the graph G is not connected, the goal of the problem is to find a *minimum spanning forest*, which is defined to be a minimum spanning tree in each connected component of G. Both problems will be referred to as the *MST* problem.

The randomized MST algorithm by Karger, Klein and Tarjan [9] which is considered here will be called the *KKT algorithm*. Also it will be assumed that the input graph $G = (V, E)$ has n vertices and m edges, and that the edge-weights are distinct.

The MST problem has been studied extensively prior to the KKT result, and several very efficient, deterministic algorithms are available from these studies. All of these are deterministic and are based on a method that greedily adds an edge to a forest that is a subgraph of the minimum spanning tree at all times. The early algorithms in this class are already efficient with a running time of $O(m\log n)$. These include the algorithms of Borůvka [1], Jarník [8] (later rediscovered by Dijkstra and Prim [5]) and Kruskal [5].

The fastest algorithm known for MST prior to the KKT algorithm runs in time $O(m\log\beta(m,n))$ [7], where $\beta(m,n) = \min\{i \mid \log^{(i)} n \le m/n\}$ [7]; here $\log^{(i)} n$ is defined as $\log n$ if $i = 1$ and as $\log\log^{(i-1)} n$ if $i > 1$. Although this running time is close to linear, it is not linear-time if the graph is very sparse.

The problem of finding the minimum spanning tree efficiently is an important and fundamental problem in graph algorithms and combinatorial optimization.

Background

Some relevant background is summarized here.

- The basic step in Borůvka's algorithm [1] is the *Borůvka step*, which picks the minimum weight edge incident on each vertex, adds it to the minimum spanning tree, and then contracts these edges. This step runs in linear time, and also very efficiently in parallel. It is the backbone of most efficient parallel algorithms for minimum spanning tree, and is also used in the KKT algorithm.

- A related and simpler problem is that of *minimum spanning tree verification*. Here, given a spanning tree T of the input edge-weighted graph, one needs to determine if T is its minimum spanning tree. An algorithm that solves this problem with a linear number of edge-weight comparisons was shown by Komlós [13], and later a deterministic linear-time algorithm was given in [6] (see also [12] for a simpler algorithm).

Key Results

The main result in [9] is a randomized algorithm for the minimum spanning tree problem that runs in expected linear time. The only operations performed on the edge-weights are pairwise comparisons. The algorithm does not assume any particular representation of the edge-weights (i. e., integer or real values), and only assumes that any comparison between a pair of edge-weights can be performed in unit time. The paper also shows that the algorithm runs in $O(m+n)$ time with the exponentially high probability $1-exp(-\Omega(m))$, and that its worst-case running time is $O(n+m\log n)$.

The simple and elegant *MST sampling lemma* given in Lemma 1 below is the key tool used to derive and analyze the KKT algorithm. This lemma needs a couple of definitions and facts:

1. The well-known *cycle property* for minimum spanning tree states that the heaviest edge in any cycle in the input graph G *cannot* be in the minimum spanning tree.
2. Let F be a forest of G (i. e., an acyclic subgraph of G). An edge $e \in E$ is *F-light* if $F \cup \{e\}$ either continues to be a forest of G, or the heaviest edge in the cycle containing e is *not* e. An edge in G that is not F-light is *F-heavy*. Note that by the cycle property, an F-heavy edge cannot be in the minimum spanning tree of G, *no matter what forest F is used*. Given a forest F of G, the set of F-heavy edges can be determined in linear time by a simple modification to existing linear-time minimum spanning tree verification algorithms [6,12].

Lemma 1 (MST Sampling Lemma) *Let $H=(V,E_H)$ be formed from the input edge-weighted graph $G=(V,E)$ by including each edge with probability p independent of the other edges. Let F be the minimum spanning forest of H. Then, the expected number of F-light edges in G is $\leq n/p$.*

The KKT algorithm identifies edges in the minimum spanning tree of G only using Borůvka steps. However, after every two Borůvka steps, it removes F-heavy edges using the minimum spanning forest F of a subgraph obtained through sampling edges with probability $p=1/2$. As mentioned earlier, these F-heavy edges can be identified in linear time. The minimum spanning forest of the sampled graph is computed recursively.

The correctness of the KKT algorithm is immediate since every F-heavy edge it removes cannot be in the MST of G since F is a forest of G, and every edge it adds to the minimum spanning tree is in the MST since it is added through a Borůvka step.

The expected running time analysis as well as the exponentially high probability bound for the running time are surprisingly simple to derive using the MST Sampling Lemma (Lemma 1).

In summary, the paper [9] proves the following results.

Theorem 2 *The KKT algorithm is a randomized algorithm that finds a minimum spanning tree of an edge-weighted undirected graph on n nodes and m edges in $O(n+m)$ time with probability at least $1-\exp(-\Omega(m))$. The expected running time is $O(n+m)$ and the worst-case running time is $O(n+m\log n)$.*

The model of computation used in [9] is the unit-cost RAM model since the known MST verification algorithms were for this model, and not the more restrictive *pointer machine* model. More recently the MST verification result and hence the KKT algorithm have been shown to work on the pointer machine as well [2].

Lemma 1 is proved in [9] through a simulation of Kruskal's algorithm along with an analysis of the probability with which an F-light edge is not sampled. Another proof that uses a backward analysis is given in [3].

Further Comments

- Recently (and since the appearance of the KKT algorithm in 1995), two new deterministic algorithms for MST have appeared, due to Chazelle [4] and Pettie and Ramachandran [14]. The former [4] runs in $O(n+m\alpha(m,n))$ time, where α is an inverse of the Ackermann's function, whose growth rate is even smaller than the β function mentioned earlier for the best result that was known prior to the KKT algorithm [7]. The latter algorithm [14] provably runs in time that is within a constant factor of the decision-tree complexity of the MST problem, and hence is optimal; its time bound is $O(n+m\alpha(m,n))$ and $\Omega(n+m)$, and the exact bound remains to be determined.
- Although the KKT algorithm runs in expected linear time (and with exponentially high probability), it is not the last word on randomized MST algorithms. A randomized MST algorithm that runs in expected linear

time and uses only $O(\log^* n)$ random bits is given in [16,17]. In contrast, the KKT algorithm uses a linear number of random bits.

Applications

The minimum spanning tree problems has a large number of applications, which are discussed in Minimum spanning trees.

Open Problems

Some open problems that remain are the following:

1. Can randomness be removed in the KKT algorithm? A hybrid algorithm that uses the KKT algorithm within a modified version of the Pettie–Ramachandran algorithm [14] is given in [16,17] that achieves expected linear time while reducing the number of random bits used to only $O(\log^* n)$. Can this tiny amount of randomness be removed as well? If all randomness can be removed from the KKT algorithm, that will establish a linear time bound for the Pettie–Ramachandran algorithm [14] and also provide another optimal deterministic MST algorithm, this one based on the KKT approach.
2. Can randomness be removed from the work-optimal *parallel algorithms* [10] for MST? A linear-work, expected logarithmic-time parallel MST algorithm for the EREW PRAM is given in [15]. This parallel algorithm is both work- and time-optimal. However, it uses a linear number of random bits. Another work-optimal parallel algorithm is given in [16,17] that runs in expected polylog time using only polylog random bits. This leads to the following open questions regarding parallel algorithms for the MST problem:
   - To what extent can dependence on random bits be reduced (from the current linear bound) in a time- and work-optimal parallel algorithm for MST?
   - To what extent can the dependence on random bits be reduced (from the current polylog bound) in a work-optimal parallel algorithm with reasonable parallelism (say polylog parallel time)?

Experimental Results

Katriel, Sanders, and Träff [11] performed an experimental evaluation of the KKT algorithm and showed that it has good performance on moderately dense graphs.

Cross References

► Minimum Spanning Trees

Acknowledgments

This work was supported in part by NSF grant CFF-0514876.

Recommended Reading

1. Borůvka, O.: O jistém problému minimálním. Práce Moravské Přírodovědecké Společnosti **3**, 37–58 (1926) (In Czech)
2. Buchsbaum, A., Kaplan, H., Rogers, A., Westbrook, J.R.: Linear-time pointer-machine algorithms for least common ancestors, MST verification and dominators. In: Proc. ACM Symp. on Theory of Computing (STOC), 1998, pp. 279–288
3. Chan, T.M.: Backward analysis of the Karger–Klein–Tarjan algorithm for minimum spanning trees. Inf. Process. Lett. **67**, 303–304 (1998)
4. Chazelle, B.: A minimum spanning tree algorithm with inverse-Ackermann type complexity. J. ACM **47**(6), 1028–1047 (2000)
5. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms. MIT Press, Cambridge (2001)
6. Dixon, B., Rauch, M., Tarjan, R.E.: Verification and sensitivity analysis of minimum spanning trees in linear time. SIAM J. Comput. **21**(6), 1184–1192 (1992)
7. Gabow, H.N., Galil, Z., Spencer, T.H., Tarjan, R.E.: Efficient algorithms for finding minimum spanning trees in undirected and directed graphs. Comb. **6**, 109–122 (1986)
8. Graham, R.L., Hell, P.: On the history of the minimum spanning tree problem. Ann. Hist. Comput. **7**(1), 43–57 (1985)
9. Karger, D.R., Klein, P.N., Tarjan, R.E.: A randomized linear-time algorithm for finding minimum spanning trees. J. ACM **42**(2), 321–329 (1995)
10. Karp, R.M., Ramachandran, V.: Parallel algorithms for shared-memory machines. In: van Leeuwen, J. (ed.) Handbook of Theoretical Computer Science, pp. 869–941. Elsevier Science Publishers B.V., Amsterdam (1990)
11. Katriel, I., Sanders, P., Träff, J.L.: A practical minimum spanning tree algorithm using the cycle property. In: Proc. 11th Annual European Symposium on Algorithms. LNCS, vol. 2832, pp. 679–690. Springer, Berlin (2003)
12. King, V.: A simpler minimum spanning tree verification algorithm. Algorithmica **18**(2), 263–270 (1997)
13. Komlós, J.: Linear verification for spanning trees. Combinatorica **5**(1), 57–65 (1985)
14. Pettie, S., Ramachandran, V.: An optimal minimum spanning tree algorithm. J. ACM **49**(1), 16–34 (2002)
15. Pettie, S., Ramachandran, V.: A randomized time-work optimal parallel algorithm for finding a minimum spanning forest. SIAM J. Comput. **31**(6), 1879–1895 (2002)
16. Pettie, S., Ramachandran, V.: Minimizing randomness in minimum spanning tree, parallel connectivity, and set maxima algorithms. In: Proc. ACM-SIAM Symp. on Discrete Algorithms (SODA), 2002, pp. 713–722
17. Pettie, S., Ramachandran, V.: New randomized minimum spanning tree algorithms using exponentially fewer random bits. ACM Trans. Algorithms. **4**(1), article 5 (2008)

Randomized Parallel Approximations to Max Flow

1991; Serna, Spirakis

Maria Serna
Department of Language & System Information,
Technical University of Catalonia, Barcelona, Spain

Keywords and Synonyms

Approximate maximum flow construction

Problem Definition

The work of Serna and Spirakis provides a parallel approximation schema for the Maximum Flow problem. An approximate algorithm provides a solution whose cost is within a factor of the optimal solution. The notation and definitions are the standard ones for networks and flows (see for example [2,7]).

A *network* $N = (G, s, t, c)$ is a structure consisting of a directed graph $G = (V, E)$, two distinguished vertices, $s, t \in V$ (called the *source* and the *sink*), and $c : E \to \mathbb{Z}^+$, an assignment of an integer capacity to each edge in E. A *flow function* f is an assignment of a non-negative number to each edge of G (called the flow into the edge) such that first at no edge does the flow exceed the capacity, and second for every vertex except s and t, the sum of the flows on its incoming edges equals the sum of the flows on its outgoing edges. The *total flow* of a given flow function f is defined as the net sum of flow into the sink t. The Maximum Flow problem can be stated as

Name	Maximum Flow
Input	A network $N = (G, s, t, c)$
Output	Find a flow f for N for which the total flow is maximum.

Maximum Flows and Matchings The Maximum Flow problem is closely related to the Maximum Matching problem on bipartite graphs.

Given a graph $G = (V, E)$ and a set of edges $M \subseteq E$ is a *matching* if in the subgraph (V, M) all vertices have degree at most one. A *maximum matching* for G is a matching with a maximum number of edges. For a graph $G = (V, E)$ with weight $w(e)$, the *weight* of a matching M is the sum of the weights of the edges in M. The problem can be stated as follows:

Name	Maximum Weight Matching
Input	A graph $G = (V, E)$ and a weight $w(e)$ for each edge $e \in E$
Output	Find a matching of G with the maximum possible weight.

There is a standard reduction from the Maximum Matching problem for bipartite graphs to the Maximum Flow problem ([7,8]). In the general weighted case one has just to look at each edge with capacity $c > 1$ as c edges joining the same points each with capacity one, and transform the multigraph obtained as shown before. Notice that to perform this transformation a c value is required which is polynomially bounded. The whole procedure was introduced by Karp, Upfal, and Wigderson [5] providing the following results

Theorem 1 *The Maximum Matching problem for bipartite graphs is NC equivalent to the Maximum Flow problem on networks with polynomial capacities. Therefore, the Maximum Flow with polynomial capacities problem belongs to the class RNC.*

Key Results

The first contribution is an extension of Theorem 1 to a generalization of the problem, namely the Maximum Flow on networks with polynomially bounded maximum flow. The proof is based on the construction (in NC) of a second network which has the same maximum flow but for which the maximum flow and the maximum capacity in the network are polynomially related.

Lemma 2 *Let $N = (G, s, t, c)$. Given any integer k, there is an NC algorithm that decides whether $f(N) \geq k$ or $f(N) < km$.*

Since Lemma 2 applies even to numbers that are exponential in size, they get

Lemma 3 *Let $N = (G, s, t, c)$ be a network, there is an NC algorithm that computes an integer value k such that $2^k \leq f(N) < m\, 2^{k+1}$.*

The following lemma establishes the NC-reduction from the Maximum Flow problem with polynomial maximum flow to the Maximum Flow problem with polynomial capacities.

Lemma 4 *Let $N = (G, s, t, c)$ be a network, there is an NC algorithm that constructs a second network $N_1 = (G, s, t, c_1)$ such that*

$$\log(\mathrm{Max}(N_1)) \leq \log(f(N_1)) + O(\log n)$$

and $f(N) = f(N_1)$.

Lemma 4 shows that the Maximum Flow problem restricted to networks with polynomially bounded maximum flow is NC-reducible to the Maximum Flow problem restricted to polynomially bounded capacities, the latter problem is a simplification of the former one, so the following results follow.

Theorem 5 *For each polynomial p, the problem of constructing a maximum flow in a network N such that $f(N) \leq p(n)$ is NC-equivalent to the problem of constructing a maximum matching in a bipartite graph, and thus it is in RNC.*

Recall that [5] gave us an $O(\log^2 n)$ randomized parallel time algorithm to compute a maximum matching. The combination of this with the reduction from the Maximum Flow problem to the Maximum Matching leads to the following result.

Theorem 6 *There is a randomized parallel algorithm to construct a maximum flow in a directed network, such that the number of processors is bounded by a polynomial in the number of vertices and the time used is $O((\log n)^\alpha \log f(N))$ for some constant $\alpha > 0$.*

The previous theorem is the first step towards finding an approximate maximum flow in a network N by an RNC algorithm. The algorithm, given N and an $\varepsilon > 0$, outputs a solution f' such that $f(N)/f' \leq 1 + 1/\varepsilon$. The algorithm uses a polynomial number of processors (independent of ε) and parallel time $O(\log^\alpha n(\log n + \log \varepsilon))$, where α is independent of ε. Thus, the algorithm is an RNC one as long as ε is at most polynomial in n. (Actually ε can be $O(n^{\log^\beta n})$ for some β.) Thus, being a Fully RNC approximation scheme (FRNCAS).

The second ingredient is a rough NC approximation to the Maximum Flow problem.

Lemma 7 *Let $N = (G, s, t, c)$ be a network. Let $k \geq 1$ be an integer, then there is an NC algorithm to construct a network $M = (G, s, t, c_1)$ such that $k f(M) \leq f(N) \leq k f(M) + km$.*

Putting all together and allowing randomization the algorithm can be sketched as follows:

```
FAST-FLOW($N = (G, s, t, c), \varepsilon$)
1. Compute $k$ such that $2^k \leq F(N) \leq 2^{k+1} m$.
2. Construct a network $N_1$ such that
       $\log(\mathrm{Max}(N_1)) \leq \log(F(N_1)) + O(\log n)$ .
3. If $2^k \leq (1+\varepsilon)m$ then $F(N) \leq (1+\varepsilon)m^2$ so use the algorithm given in Theorem 6 to solve the Maximum Flow problem in $N$ as a Maximum Matching and **return**
4. Let $\beta = \lfloor (2^k)/((1+\varepsilon)m) \rfloor$. Construct $N_2$ from $N_1$ and $\beta$ using the construction in Lemma 7.
5. Solve the Maximum Flow problem in $N_2$ as a Maximum Matching.
6. Output $F' = \beta F(M_2)$ and for all $e \in E$, $f'(e) = \beta f(e)$.
```

Theorem 8 *Let $N = (G, s, t, c)$ be a network. Then, algorithm FAST-FLOW is an RNC algorithm such that for all $\varepsilon > 0$ at most polynomial in the number of network vertices, the algorithm computes a legal flow of value f' such that*

$$\frac{f(N)}{f'} \leq 1 + \frac{1}{\varepsilon} .$$

Furthermore, the algorithm uses a polynomial number of processors and runs in expected parallel time $O(\log^\alpha n(\log n + \log \varepsilon))$, for some constant α, independent of ε.

Applications

The *rounding/scaling* technique is used in general to deal with problems that are hard due to the presence of large weights in the problem instance. The technique modifies the problem instance in order to produce a second instance that has no large weights, and thus can be solved efficiently. The way in which a new instance is obtained consists of computing first an estimate of the optimal value (when needed) in order to discard unnecessary high weights. Then the weights are modified, scaling them down by an appropriate factor that depends on the estimation and the allowed error. The rounding factor is determined in such a way that the so-obtained instance can be solved efficiently. Finally, a last step consisting of scaling up the value of the "easy" instance solution is performed in order to meet the corresponding accuracy requirements.

It is known that in the sequential case, the only way to construct FPTAS uses rounding/scaling and interval partition [6]. In general, both techniques can be paralyzed, although sometimes the details of the parallelization are non-trivial [1].

The Maximum Flow problem has a long history in Computer Science. Here are recorded some results about its parallel complexity. Goldschlager, Shaw, and Staples showed that the Maximum Flow problem is P-complete [3]. The P-completeness proof for Maximum Flow uses large capacities on the edges; in fact the values of some capacities are exponential in the number of network vertices. If the capacities are constrained to be no greater than some polynomial in the number of network vertices the problem is in ZNC. In the case of planar networks it is known that the Maximum Flow problem is in NC, even if arbitrary capacities are allowed [4].

Open Problems

The parallel complexity of the Maximum Weight Matching problem when the weight of the edges are given in binary is still an open problem. However, as mentioned earlier, there is a randomized NC algorithm to solve the problem in $O(\log^2 n)$ parallel steps, when the weights of the edges are given in unary. The scaling technique has been used to obtain fully randomized NC approximation schemes, for the Maximum Flow and Maximum Weight Matching problems (see [10]). The result appears to be the best possible in regard of full approximation, in the sense that the existence of an FNCAS for any of the problems considered is equivalent to the existence of an NC algorithm for perfect matching which is also still an open problem.

Cross References

► Approximate Maximum Flow Construction
► Maximum Matching
► Paging

Recommended Reading

1. Díaz, J., Serna, M., Spirakis, P.G., Torán, J.: Paradigms for fast parallel approximation. In: Cambridge International Series on Parallel Computation, vol 8, Cambridge University Press, Cambridge (1997)
2. Even, S.: Graph Algorithms. Computer Science Press, Potomac (1979)
3. Goldschlager, L.M., Shaw, R.A., Staples, J.: The maximum flow problem is log-space complete for P. Theor. Comput. Sci. **21**, 105–111 (1982)
4. Johnson, D.B., Venkatesan, S.M.: Parallel algorithms for minimum cuts and maximum flows in planar networks. J. ACM **34**, 950–967 (1987)
5. Karp, R.M., Upfal, E., Wigderson, A.: Constructing a perfect matching is in Random NC. Combin. **6**, 35–48 (1986)
6. Korte, B., Schrader, R.: On the existence of fast approximation schemes. Nonlinear Program. **4**, 415–437 (1980)
7. Lawler, E.L.: Combinatorial Optimization: Networks and Matroids. Holt, Rinehart and Winston, New York (1976)
8. Papadimitriou, C.: Computational Complexity. Addison-Wesley, Reading (1994)
9. Peters, J.G., Rudolph, L.: Parallel aproximation schemes for subset sum and knapsack problems. Acta Inform. **24**, 417–432 (1987)
10. Spirakis, P.: PRAM models and fundamental parallel algorithm techniques: Part II. In: Gibbons, A., Spirakis, P. (eds.) Lectures on Parallel Computation, pp. 41–66. Cambrige University Press, New York (1993)

Randomized Rounding

1987; Raghavan, Thompson

Rajmohan Rajaraman
Department of Computer Science,
Northeastern University,
Boston, MA, USA

Problem Definition

Randomized rounding is a technique for designing approximation algorithms for NP-hard optimization problems. Many combinatorial optimization problems can be represented as 0-1 integer linear programs; that is, integer linear programs in which variables take values in $\{0,1\}$. While 0-1 integer linear programming is NP-hard, the rational relaxations (also referred to as fractional relaxations) of these linear programs are solvable in polynomial time [12,13]. Randomized rounding is a technique to construct a provably good solution to a 0-1 integer linear program from an optimum solution to its rational relaxation by means of a randomized algorithm.

Let Π be a 0-1 integer linear program with variables $x_i \in \{0,1\}$, $1 \le i \le n$. Let Π_R be the rational relaxation of Π obtained by replacing the $x_i \in \{0,1\}$ constraints by $x_i \in [0,1]$, $1 \le i \le n$. The randomized rounding approach consists of two phases:

1. Solve Π_R using an efficient linear program solver. Let the variable x_i take on value $x_i^* \in [0,1]$, $1 \le i \le n$.
2. Compute a solution to Π by setting the variables x_i randomly to one or zero according to the following rule:

$$\Pr[x_i = 1] = x_i^* \, .$$

For several fundamental combinatorial optimization problems, the randomized rounding technique yields simple randomized approximation algorithms that yield solutions provably close to optimal. Variants of the basic approach outlined above, in which the rounding of variable x_i in the second phase is done with a probability that is some appropriate function of x_i^*, have also been studied. The analyses of algorithms based on randomized rounding often rely on Chernoff–Hoeffding bounds from probability theory [5,11].

The work of Raghavan and Thompson [14] introduced the technique of randomized rounding for designing approximation algorithms for NP-hard optimization problems. The randomized rounding approach also implicitly proves the existence of a solution with certain desirable properties. In this sense, randomized rounding can be viewed as a variant of the probabilistic method, due to Erdös [1], which is widely used for various existence proofs in combinatorics.

Raghavan and Thompson illustrate the randomized rounding approach using three optimization problems: VLSI routing, multicommodity flow, and k-matching in hypergraphs.

Definition 1 In the **VLSI Routing** problem, we are given a two-dimensional rectilinear lattice L_n over n nodes and a collection of m *nets* $\{a_i : 1 \le i \le m\}$, where net a_i, is a set of nodes to be connected by means of a Steiner tree in L_n. For each net a_i, we are also given a set $\mathcal{A}_i$ of *allowed* trees that can be used for connecting the nodes in that set. A solution to the problem is a set $\mathcal{T}$ of trees $\{T_i \in \mathcal{A}_i : 1 \le i \le m\}$. The *width* of solution $\mathcal{T}$ is the maximum, over all edges e, of the number of trees in $\mathcal{T}$

that contain the edge. The goal of the VLSI routing problem is to determine a solution with minimum width.

Definition 2 In the **Multicommodity Flow Congestion Minimization** problem (or simply, the Congestion Minimization problem), we are given a graph $G = (V, E)$, and a set of source-destination pairs $\{(s_i, t_i) : 1 \leq i \leq k\}$. For each pair (s_i, t_i), we would like to route one unit of demand from s_i to t_i. A solution to the problem is a set $\mathcal{P} = \{P_i : 1 \leq i \leq k\}$ such that P_i is a path from s_i to t_i in G. We define the *congestion* of $\mathcal{P}$ to be the maximum, over all edges e, of the number of paths containing e. The goal of the undirected multicommodity flow problem is to determine a path set $\mathcal{P}$ with minimum congestion.

In their original work [14], Raghavan and Thompson studied the above problem for the case of undirected graphs and referred to it as the Undirected Multicommodity Flow problem. Here, we adopt the more commonly-used term of Congestion Minimization and consider both undirected and directed graphs since the results of [14] apply to both classes of graphs. Researchers have studied a number of variants of the multicommodity flow problem, which differ in various aspects of the problem such as the nature of demands (e. g., uniform vs. non-uniform), the objective function (e. g., the total flow vs. the maximum fraction of each demand), and edge capacities (e. g., uniform vs. non-uniform).

Definition 3 In the **Hypergraph Simple k-Matching** problem, we are given a hypergraph H over an n-element vertex set V. A k-matching of H is a set M of edges such that each vertex in V belongs to at most k of the edges in M. A k-matching M is simple if no edge in H occurs more than once in M. The goal of the problem is to determine a maximum-size simple k-matching of a given hypergraph H.

Key Results

Raghavan and Thompson present approximation algorithms for the above three problems using randomized rounding. In each case, the algorithm is easy to present: write a 0-1 integer linear program for the problem, solve the rational relaxation of this program, and then apply randomized rounding. They establish bounds on the quality of the solutions (i. e., the approximation ratios of the algorithm) using Chernoff–Hoeffding bounds on the tail of the sums of bounded and independent random variables [5,11].

The VLSI Routing problem can be easily expressed as a 0-1 integer linear program, say Π_1. Let W^* denote the width of the optimum solution to the rational relaxation of Π_1.

Theorem 1 *For any ε such that $0 < \varepsilon < 1$, the width of the solution produced by randomized rounding does not exceed*

$$W^* + \left[3W^* \ln \frac{2n(n-1)}{\varepsilon}\right]^{1/2}$$

with probability at least $1 - \varepsilon$, provided $W^* \geq 3\ln(2n(n-1)/\varepsilon)$.

Since W^* is a lower bound on the width of an optimum solution to Π_1, it follows that the randomized rounding algorithm has an approximation ratio of $1 + o(1)$ with high probability as long as W^* is sufficiently large.

The Congestion Minimization problem can be easily expressed as a 0-1 integer linear program, say Π_2. Let C^* denote the congestion of the optimum solution to the linear relaxation of Π_2. This optimum solution yields a set of flows, one for each commodity i. The flow for commodity i can be decomposed into a set Γ_i of at most $|E|$ paths from s_i to t_i. The randomized rounding algorithm selects, for each commodity i, one path P_i at random from Γ_i according to the flow values determined by the flow decomposition.

Theorem 2 *For any ε such that $0 < \varepsilon < 1$, the capacity of the solution produced by randomized rounding does not exceed*

$$C^* + \left[3C^* \ln \frac{|E|}{\varepsilon}\right]^{1/2}$$

with probability at least $1 - \varepsilon$, provided $C^* \geq 2\ln|E|$.

Since C^* is a lower bound on the width of an optimum solution to Π_1, it follows that the randomized rounding algorithm achieves a constant approximation ratio with probability $1 - 1/n$ when C^* is $\Omega(\log n)$.

For both the VLSI Routing and the Congestion Minimization problems, slightly worse approximation ratios are achieved if the lower bound condition on W^* and C^*, respectively, is removed. In particular, the approximation ratio achieved is $O(\log n/\log\log n)$ with probability at least $1 - n^{-c}$ for a constant $c > 0$ whose value depends on the constant hidden in the big-Oh notation.

The hypergraph k-matching problem is different than the above two problems in that it is a packing problem with a maximization objective while the latter are covering problems with a minimization objective. Raghavan and Thompson show that randomization rounding, in conjunction with a scaling technique, yields good approximation algorithms for the hypergraph k-matching problem.

They first express the matching problem as a 0-1 integer linear program, solve its rational relaxation Π_3, and then round the optimum rational solution by using appropriately scaled values of the variables as probabilities. Let S^* denote the value of the optimum solution to Π_3.

Theorem 3 *Let δ_1 and δ_2 be positive constants such that $\delta_2 > n \cdot \mathrm{e}^{-k/6}$ and $\delta_1 + \delta_2 < 1$. Let $\alpha = 3 \ln(n/\delta_2)/k$ and*

$$S' = S^* \left(1 - \frac{(\alpha^2 + 4\alpha)^{1/2} - \alpha}{2}\right) .$$

Then, there exists a simple k-matching for the given hypergraph with size at least

$$S' - \left(2S' \ln \frac{1}{\delta_1}\right)^{1/2} .$$

Note that the above result is stated as an existence result. It can be modified to yield a randomized algorithm that achieves essentially the same bound with probability $1 - \varepsilon$ for a given failure probability ε.

Applications

Randomized rounding has found applications for a wide range of combinatorial optimization problems. Following the work of Raghavan and Thompson [14], Goemans and Williamson showed that randomized rounding yields an $e/(e-1)$-approximation algorithm for MAXSAT, the problem of finding an assignment that satisfies the maximum number of clauses of a given Boolean formula [7]. For the set cover problem, randomized rounding yields an algorithm with an asymptotically optimal approximation ratio of $O(\log n)$, where n is the number of elements in the given set cover instance [10]. Srinivasan has developed more sophisticated randomized rounding approaches for set cover and more general covering and packing problems [15]. Randomized rounding also yields good approximation algorithms for several flow and cut problems, including variants of undirected multicommodity flow [9] and the multiway cut problem [4].

While randomized rounding provides a unifying approach to obtain approximation algorithms for hard optimization problems, better approximation algorithms have been designed for specific problems. In some cases, randomized rounding has been combined with other algorithms to yield better approximation ratios than previously known. For instance, Goemans and Williamson showed that the better of two solutions, one obtained by randomized rounding and the other obtained by an earlier algorithm due to Johnson, yields a 4/3 approximation for MAXSAT [7].

The work of Raghavan and Thompson applied randomized rounding to a solution obtained for the relaxation of a 0-1 integer program for a given problem. In recent years, more sophisticated approximation algorithms have been obtained by applying randomized rounding to semidefinite program relaxations of the given problem. Examples include the 0.87856-approximation algorithm for MAXCUT due to Goemans and Williamson [8] and an $O(\sqrt{\log n})$-approximation algorithm for the sparsest cut problem, due to Arora, Rao, and Vazirani [3].

An excellent reference for the above and other applications of randomized rounding in approximation algorithms is the text by Vazirani [16].

Open Problems

While randomized rounding has yielded improved approximation algorithms for a number of NP-hard optimization problems, the best approximation achievable by a polynomial-time algorithm is still open for most of the problems discussed in this article, including MAXSAT, MAXCUT, the sparsest cut, the multiway cut, and several variants of the congestion minimization problem. For directed graphs, it has been shown that best approximation ratio achievable for congestion minimization in polynomial time is $\Omega(\log n / \log\log n)$, unless NP $\subset$ ZPTIME$(n^{O(\log\log n)})$, matching the upper bound mentioned in Sect. “Key Results” up to constant factors [6]. For undirected graphs, the best known inapproximability lower bound is $\Omega(\log\log n / \log\log\log n)$ [2].

Cross References

▶ Oblivious Routing

Recommended Reading

1. Alon, N., Spencer, J.H.: The Probabilistic Method. Wiley, New York (1991)
2. Andrews, M., Zhang, L.: Hardness of the undirected congestion minimization problem. In: STOC '05: Proceedings of the thirty-seventh annual ACM symposium on Theory of computing, pp. 284–293. ACM Press, New York (2005)
3. Arora, S., Rao, S., Vazirani, U.V.: Expander flows, geometric embeddings and graph partitioning. In: STOC, pp. 222–231. (2004)
4. Calinescu, G., Karloff, H.J., Rabani, Y.: An improved approximation algorithm for multiway cut. J. Comput. Syst. Sci. **60**(3), 564–574 (2000)
5. Chernoff, H.: A measure of the asymptotic efficiency for tests of a hypothesis based on the sum of observations. Ann. Math. Stat. **23**, 493–509 (1952)
6. Chuzhoy, J., Guruswami, V., Khanna, S., Talwar, K.: Hardness of routing with congestion in directed graphs. In: STOC '07: Pro-

ceedings of the thirty-ninth annual ACM symposium on Theory of computing, pp. 165–178. ACM Press, New York (2007)
7. Goemans, M.X., Williamson, D.P.: New 3/4-approximation algorithms for the maximum satisfiability problem. SIAM J. Discret. Math. **7**, 656–666 (1994)
8. Goemans, M.X., Williamson, D.P.: Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. J. ACM **42**(6), 1115–1145 (1995)
9. Guruswami, V., Khanna, S., Rajaraman, R., Shepherd, B., Yannakakis, M.: Near-optimal hardness results and approximation algorithms for edge-disjoint paths and related problems. J. Comput. Syst. Sci. **67**, 473–496 (2003)
10. Hochbaum, D.S.: Approximation algorithms for the set covering and vertex cover problems. SIAM J. Comput. **11**(3), 555–556 (1982)
11. Hoeffding, W.: On the distribution of the number of successes in independent trials. Ann. Math. Stat. **27**, 713–721 (1956)
12. Karmarkar, N.: A new polynomial-time algorithm for linear programming. Combinatorica **4**, 373–395 (1984)
13. Khachiyan, L.G.: A polynomial algorithm for linear programming. Soviet Math. Doklady **20**, 191–194 (1979)
14. Raghavan, P., Thompson, C.: Randomized rounding: A technique for provably good algorithms and algorithmic proofs. Combinatorica **7** (1987)
15. Srinivasan, A.: Improved approximations of packing and covering problems. In: Proceedings of the 27th Annual ACM Symposium on Theory of Computing, pp. 268–276 (1995)
16. Vazirani, V.: Approximation Algorithms. Springer (2003)

Randomized Searching on Rays or the Line

1993; Kao, Reif, Tate

Stephen R. Tate
University of North Carolina at Greensboro, Greensboro, NC, USA

Keywords and Synonyms

Cow-path problem; On-line navigation

Problem Definition

This problem deals with finding a point at an unknown position on one of a set of w rays which extend from a common point (the origin). In this problem there is a *searcher*, who starts at the origin, and follows a sequence of commands such as "explore to distance d on ray i." The searcher detects immdiately when the target point is crossed, but there is no other information provided from the search environment. The goal of the searcher is to minimize the distance traveled.

There are several different ways this problem has been formulated in the literature, including one called the "cow-path problem" that involves a cow searching for a pasture down a set of paths. When $w = 2$, this problem is to search for a point on the line, which has also been described as a robot searching for a door in an infinite wall or a shipwreck survivor searching for a stream after washing ashore on a beach.

Notation

The problem is as described above, with w rays. The position of the target point (or goal) is denoted (g, i) if it is at distance g on ray $i \in \{0, 1, \cdots, w-1\}$. The standard notion of *competitive ratio* is used when analyzing algorithms for this problem: An algorithm that knows which ray the goal is on will simply travel distance g down that ray before stopping, so search algorithms are compared to this optimal, omniscient strategy.

In particular, if $\mathcal{R}$ is a randomized algorithm, then the distance traveled to find a particular goal position is a random variable denoted $\mathsf{distance}(\mathcal{R}, (g, i))$, with expected value $E[\mathsf{distance}(\mathcal{R}, (g, i))]$. Algorithm $\mathcal{R}$ has competitive ratio c if there is a constant a such that, for all goal positions (g, i),

$$E[\mathsf{distance}(\mathcal{R}, (g, i))] \le c \cdot g + a\,. \qquad (1)$$

Key Results

This problem is solved optimally using a randomized geometric sweep strategy: Search through the rays in a random (but fixed) order, with each search distance a constant factor longer than the preceding one. The initial search distance is picked from a carefully selected probability distribution, giving the following algorithm:

```
RaySearch$_{r,w}$
  $\sigma \leftarrow$ A random permutation of $\{0, 1, 2, \cdots, w-1\}$;
  $\epsilon \leftarrow$ A random real uniformly chosen from $[0, 1)$;
  $d \leftarrow r^{\epsilon}$;
  $p \leftarrow 0$;
  repeat
    Explore path $\sigma(p)$ up to distance $d$;
    if goal not found then return to origin;
    $d \leftarrow d \cdot r$;
    $p \leftarrow (p+1) \bmod w$;
  until goal found;
```

The theorems below give the competitive ratio of this algorithm, show how to pick the best r, and establish the optimality of the algorithm.

Theorem 1 ([9]) *For any fixed $r > 1$, Algorithm* RaySearch$_{r,w}$ *has competitive ratio*

$$R(r, w) = 1 + \frac{2}{w} \cdot \frac{1 + r + r^2 + \cdots + r^{w-1}}{\ln r}\,,$$

Randomized Searching on Rays or the Line, Table 1
The asymptotic growth of the competitive ratio with w is established in the following theorem

w	r_w^*	Optimal randomized ratio	Optimal deterministic ratio
2	3.59112	4.59112	9
3	2.01092	7.73232	14.5
4	1.62193	10.84181	19.96296
5	1.44827	13.94159	25.41406
6	1.35020	17.03709	30.85984
7	1.28726	20.13033	36.30277

Theorem 2 ([9]) *The unique solution of the equation*

$$\ln r = \frac{1 + r + r^2 + \cdots + r^{w-1}}{r + 2r^2 + 3r^3 + \cdots + (w-1)r^{w-1}} \tag{2}$$

for $r > 1$, denoted by r_w^*, gives the minimum value for $R(r, w)$.

Theorem 3 ([7,9,12]) *The optimal competitive ratio for any randomized algorithm for searching on w rays is*

$$\min_{r>1} \left\{ 1 + \frac{2}{w} \cdot \frac{1 + r + r^2 + \cdots + r^{w-1}}{\ln r} \right\} .$$

Corollary 1 *Algorithm* RAYSEARCH$_{r,w}$ *is optimally competitive.*

Using Theorem 2 and standard numerical techniques, r_w^* can be computed to any required degree of precision. The following table shows, for small values of w, approximate values for r_w^* and the corresponding optimal competitive ratio (achieved by RAYSEARCH$_{r,w}$)—the optimal deterministic competitive ratio (see [1]) is also shown for comparison:

Theorem 4 ([9]) *The competitive ratio for algorithm* RAYSEARCH$_{r,w}$ *(with $r = r_w^*$) is $\kappa w + o(w)$, where*

$$\kappa = \min_{s>0} \left[2 \frac{e^s - 1}{s^2} \right] \approx 3.088 .$$

Applications

The most direct applications of this problem are in geometric searching, such as robot navigation problems. For example, when a robot is traveling in an unknown area and encounters an obstacle, a typical first step is to find the nearest corner to go around [2,3], which is just an instance of the ray searching problem (with $w = 2$).

In addition, any abstract search problem with a cost function that is linear in the distance to the goal reduces to ray searching. This includes applications in artificial intelligence that search for a goal in a largely unknown search space [11] and the construction of hybrid algorithms [7]. In hybrid algorithms, a set of algorithms $A_1, A_2, \cdots, A_w$ for solving a problem is considered—algorithm A_1 is run for a certain amount of time, and if the algorithm is not successful algorithm A_1 is stopped and algorithm A_2 is started, repeating through all algorithms as many times as is necessary to find a solution. This notion of hybrid algorithms has been used successfully for several problems (such as the first competitive algorithm for the online k-server problem [4]), and the ray search algorithm gives the optimal strategy for selecting the trial running times of each algorithm.

Open Problems

Several natural extensions of this problem have been studied in both deterministic and randomized settings, including ray-searching when an upper bound on the distance to the goal is known (i. e., the rays are not infinite, but are line segments) [10,5,12], or when a probability distribution of goal positions is known [8]. Other variations of this basic searching problem have been studied for deterministic algorithms only, such as when the searcher's control is imperfect (so distances can't be specified precisely) [6] and for more general search spaces like points in the plane [1]. A thorough study of these variants with randomized algorithms remains an open problem.

Cross References

- ▶ Alternative Performance Measures in Online Algorithms
- ▶ Deterministic Searching on the Line
- ▶ Robotics

Recommended Reading

1. Baeza-Yates, R.A., Culberson, J.C., Rawlins, G.J.E.: Searching in the plane. Inf. Comput. **16**, 234–252 (1993)
2. Berman, P., Blum, A., Fiat, A., Karloff, H., Rosén, A., Saks, M.: Randomized robot navigation algorithms. In: Proceedings, Seventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 75–84 (1996)
3. Blum, A., Raghavan, P., Schieber, B.: Navigating in unfamiliar geometric terrain. In: Proceedings 23rd ACM Symposium on Theory of Computing (STOC), pp. 494–504 (1991)
4. Fiat, A., Rabani, Y., Ravid, Y.: Competitive k-server algorithms. In: Proceedings 31st IEEE Symposium on Foundations of Computer Science (FOCS), pp. 454–463 (1990)
5. Hipke, C., Icking, C., Klein, R., Langetepe, E.: How to find a point on a line within a fixed distance. Discret. Appl. Math. **93**, 67–73 (1999)

6. Kamphans, T., Langetepe, E.: Optimal competitive online ray search with an error-prone robot. In: 4th International Workshop on Experimental and Efficient Algorithms, pp. 593–596 (2005)
7. Kao, M., Ma, Y., Sipser, M., Yin, Y.: Optimal constructions of hybrid algorithms. In: Proceedings 5th ACM-SIAM Symposium on Discrete Algorithms (SODA) pp. 372–381 (1994)
8. Kao, M.-Y., Littman, M.L.: Algorithms for informed cows. In: AAAI-97 Workshop on On-Line Search, pp. 55–61 (1997)
9. Kao, M.-Y., Reif, J.H., Tate, S.R.: Searching in an unknown environment: An optimal randomized algorithm for the cow-path problem. Inf. Comput. **133**, 63–80 (1996)
10. López-Ortiz, A., Schuierer, S.: The ultimate strategy to search on m rays? Theor. Comput. Sci. **261**, 267–295 (2001)
11. Pearl, J.: Heuristics: Intelligent Search Strategies for Computer Problem Solving. Addison-Wesley, Reading, MA (1984)
12. Schuierer, S.: A lower bound for randomized searching on m rays. In: Computer Science in Perspective, pp. 264–277 (2003)

Random Number Generation

► Weighted Random Sampling

Random Planted 3-SAT

2003; Flaxman

ABRAHAM FLAXMAN
Theory Group, Microsoft Research, Redmond, WA, USA

Keywords and Synonyms

Constraint satisfaction

Problem Definition

This classic problem in complexity theory is concerned with efficiently finding a satisfying assignment to a propositional formula. The input is a formula with n Boolean variables which is expressed as an AND of ORs with 3 variables in each OR clause (a *3-CNF formula*). The goal is to (1) find an assignment of variables to TRUE and FALSE so that the formula has value TRUE, or (2) prove that no such assignment exists. Historically, recognizing satisfiable 3-CNF formulas was the first "natural" example of an NP-complete problem, and, because it is NP-complete, no polynomial-time algorithm can succeed on all 3-CNF formulas unless P = NP [4,10]. Because of the numerous practical applications of 3-SAT, and also due to its position as the canonical NP-complete problem, many heuristic algorithms have been developed for solving 3-SAT, and some of these algorithms have been analyzed rigorously on random instances.

Notation A 3-CNF formula over variables $x_1, x_2, \ldots, x_n$ is the conjunction of m clauses $C_1 \wedge C_2 \wedge \cdots \wedge C_m$, where each clause is the disjunction of 3 literals, $C_i = \ell_{i_1} \vee \ell_{i_2} \vee \ell_{i_3}$, and each literal ℓ_{i_j} is either a variable or the negation of a variable (the negation of the variable x is denoted by $\overline{x}$). A 3-CNF formula is *satisfiable* if and only if there is an assignment of variables to truth values so that every clause contains at least one true literal. Here, all asymptotic analysis is in terms of n, the number of variables in the 3-CNF formula, and a sequence of events $\{\mathcal{E}_n\}$ is said to hold *with high probability* (abbreviated **whp**) if $\lim_{n\to\infty} \Pr[\mathcal{E}_n] = 1$.

Distributions There are many distributions over 3-CNF formulas which are interesting to consider, and this chapter focuses on dense satisfiable instances. Dense satisfiable instances can be formed by conditioning on the event $\{I_{n,m}$ is satisfiable$\}$, but this conditional distribution is difficult to sample from and to analyze. This has led to research in "planted" random instances of 3-SAT, which are formed by first choosing a truth assignment φ uniformly at random, and then selecting each clause independently from the triples of literals where at least one literal is set to TRUE by the assignment φ. The clauses can be included with equal probabilities in analogy to the $\mathbb{I}_{n,p}$ or $\mathbb{I}_{n,m}$ distributions above [8,9], or different probabilities can be assigned to the clauses with one, two, or three literals set to TRUE by φ, in an effort to better hide the satisfying assignment [2,7].

Problem 1 (3-SAT)
INPUT: *3-CNF Boolean formula $F = C_1 \wedge C_2 \wedge \cdots \wedge C_m$, where each clause C_i is of the form $C_i = \ell_{i_1} \vee \ell_{i_2} \vee \ell_{i_3}$, and each literal ℓ_{i_j} is either a variable or the negation of a variable.*
OUTPUT: *A truth assignment of variables to Boolean values which makes at least one literal in each clause TRUE, or a certificate that no such assignment exists.*

Key Results

A line of basic research dedicated to identifying hard search and decision problems, as well as the potential cryptographic applications of planted instances of 3-SAT, has motivated the development of algorithms for 3-SAT which are known to work on planted random instances.

Majority Vote Heuristic: If every clause consistent with the planted assignment is included with the same probability, then there is a bias towards including the literal satisfied by the planted assignment more frequently than its negation. This is the motivation behind the Majority Vote Heuristic, which assigns each variable to the truth value which will satisfy the majority of the clauses in which it appears. Despite its simplicity, this heuristic has been proven successful **whp** for sufficiently dense planted instances [8].

Theorem 1 *When c is a sufficiently large constant and $I \sim \mathbb{I}^{\phi}_{n,cn\log n}$, **whp** the majority vote heuristic finds the planted assignment φ.*

When the density of the planted random instance is lower than $c\log n$, then the majority vote heuristic will fail, and if the relative probability of the clauses satisfied by one, two, and three literals are adjusted appropriately then it will fail miserably. But there are alternative approaches.

For planted instances where the density is a sufficiently large constant, the majority vote heuristic provides a good starting assignment, and then the *k-OPT heuristic* can finish the job. The *k*-OPT heuristic of [6] is defined as follows: Initialize the assignment by majority vote. Initialize *k* to 1. While there exists a set of *k* variables for which flipping the values of the assignment will (1) make false clauses true and (2) will not make true clauses false, flip the values of the assignment on these variables. If this reaches a local optimum that is not a satisfying assignment, increase *k* and continue.

Theorem 2 *When c is a sufficiently large constant and $I \sim I^{\phi}_{n,cn}$ the k-OPT heuristic finds a satisfying assignment in polynomial time **whp**. The same is true even in the semi-random case, where an adversary is allowed to add clauses to I that have all three literals set to TRUE by φ before giving the instance to the k-OPT heuristic.*

A related algorithm has been shown to run in expected polynomial time in [9], and a rigorous analysis of *Warning Propagation (WP)*, a message passing algorithm related to Survey Propagation, has shown that WP is successful **whp** on planted satisfying assignments, provided that the clause density exceeds a sufficiently large constant [5].

When the relative probabilities of clauses containing one, two, and three literals are adjusted carefully, it is possible to make the majority vote assignment very different from the planted assignment. A way of setting these relative probabilities that is predicted to be difficult is discussed in [2]. If the density of these instances is high enough (and the relative probabilities are anything besides the case of "Gaussian elimination with noise"), then a spectral heuristic provides a starting assignment close to the planted assignment and local reassignment operations are sufficient to recover a satisfying assignment [7].

More formally, consider instance $I = I_{n,p_1,p_2,p_3}$, formed by choosing a truth assignment φ on n variables uniformly at random and including in I each clause with exactly i literals satisfied by φ independently with probability p_i. By setting $p_1 = p_2 = p_3$ this reduces to the distribution mentioned above.

Setting $p_1 = p_2$ and $p_3 = 0$ yields a natural distribution on 3CNFs with a planted not-all-equal assignment, a situation where the greedy variable assignment rule generates a random assignment. Setting $p_2 = p_3 = 0$ gives 3CNFs with a planted exactly-one-true assignment (which succumb to the greedy algorithm followed by the non-spectral steps below). Also, correctly adjusting the ratios of p_1, p_2, and p_3 can obtain a variety of (slightly less natural) instance distributions which thwart the greedy algorithm. Carefully selected values of p_1, p_2, and p_3 are considered in [2], where it is conjectured that no algorithm running in polynomial time can solve I_{n,p_1,p_2,p_3} **whp** when $p_i = c_i\alpha/n^2$ and

$$0.077 < c_3 < 0.25 \qquad c_2 = (1-4c_3)/6$$
$$c_1 = (1+2c_3)/6 \qquad \alpha > \frac{4.25}{7}\,.$$

The *spectral heuristic* modeled after the coloring algorithms of [1,3] was developed for such planted distributions in [7]. This polynomial time algorithm which returns a satisfying assignment to I_{n,p_1,p_2,p_3} **whp** when $p_1 = d/n^2$, $p_2 = \eta_2 d/n^2$ and $p_3 = \eta_3 d/n^2$, for $0 \le \eta_2, \eta_3 \le 1$, and $d \ge d_{\min}$, where $d_{\min}$ is a function of η_2, η_3. The algorithm is structured as follows:

1. Construct a graph G from the 3CNF.
2. Find the most negative eigenvalue of a matrix related to the adjacency matrix of G.
3. Assign a value to each variable based on the signs of the eigenvector corresponding to the most negative eigenvalue.
4. Iteratively improve the assignment.
5. Perfect the assignment by exhaustive search over a small set containing all the incorrect variables.

A more elaborate description of each step is the following:

Step (1): Given 3CNF $I = I_{n,p_1,p_2,p_3}$, where $p_1 = d/n^2$, $p_2 = \eta_2 d/n^2$, and $p_3 = \eta_3 d/n^2$, the graph in step (1), $G = (V,E)$, has $2n$ vertices, corresponding to the literals in I, and labeled $\{x_1, \overline{x}_1, \ldots x_n, \overline{x}_n\}$. G has an edge between vertices ℓ_i and ℓ_j if I includes a clause with both ℓ_i and ℓ_j (and G does not have multiple edges).

Step (2): Consider $G' = (V, E')$, formed by deleting all the edges incident to vertices with degree greater than $180d$. Let A be the adjacency matrix of G'. Let λ be the most negative eigenvalue of A and $\mathbf{v}$ be the corresponding eigenvector.

Step (3): There are two assignments to consider, π_+, which is defined by

$$\pi_+(x_i) = \begin{cases} T\,, & \text{if} \quad \mathbf{v}_i \ge 0\,; \\ F\,, & \text{otherwise}\,; \end{cases}$$

and π_-, which is defined by

$$\pi_-(x) = \neg\pi_+(x) .$$

Let π_0 be the better of π_+ and π_- (that is, the assignment which satisfies more clauses). It can be shown that π_0 agrees with φ on at least $(1 - C/d)n$ variables for some absolute constant C.

Step (4): For $i = 1, \ldots, \log n$ do the following: for each variable x, if x appears in $5\varepsilon d$ clauses unsatisfied by π_{i-1}, then set $\pi_i(x) = \neg\pi_{i-1}(x)$, where ε is an appropriately chosen constant (taking $\varepsilon = 0.1$ works); otherwise set $\pi_i(x) = \pi_{i-1}(x)$.

Step (5): Let $\pi'_0 = \pi_{\log n}$ denote the final assignment generated in step (4). Let $\mathcal{A}_4^{\pi'_0}$ be the set of variables which do not appear in $(3 \pm 4\varepsilon)d$ clauses as the only true literal with respect to assignment π'_0, and let $\mathcal{B}$ be the set of variables which do not appear in $(\mu_D \pm \varepsilon)d$ clauses, where $\mu_D d = (3+6)d + (6+3)\eta_2 d + 3\eta_3 d + \mathcal{O}(1/n)$ is the expected number of clauses containing variable x. Form partial assignment $\pi_1{}'$ by unassigning all variables in $\mathcal{A}_4^{\pi'_0}$ and $\mathcal{B}$. Now, for $i \geq 1$, if there is a variable x_i which appears in less than $(\mu_D - 2\varepsilon)d$ clauses consisting of variables that are all assigned by π'_i, then let π'_{i+1} be the partial assignment formed by unassigning x_i in π'_i. Let π' be the partial assignment when this process terminates. Consider the graph Γ with a vertex for each variable that is unassigned in π' and an edge between two variables if they appear in a clause together. If any connected component in Γ is larger than $\log n$ then fail. Otherwise, find a satisfying assignment for I by performing an exhaustive search on the variables in each connected component of Γ.

Theorem 3 *For any constants $0 \leq \eta_2, \eta_3 \leq 1$, except $(\eta_2, \eta_3) = (0, 1)$, there exists a constant $d_{\min}$ such that for any $d \geq d_{min}$, if $p_1 = d/n^2$, $p_2 = \eta_2 d/n^2$, and $p_3 = \eta_3 d/n^2$ then this polynomial-time algorithm produces a satisfying assignment for random instances drawn from I_{n,p_1,p_2,p_3}* ***whp***.

Applications

3-SAT is a universal problem, and due to its simplicity, it has potential applications in many areas, including proof theory and program checking, planning, cryptanalysis, machine learning, and modeling biological networks.

Open Problems

An important direction is to develop alternative models of random distributions which more accurately reflect the type of instances that occur in the real world.

Data Sets

Sample instances of satisfiability and 3-SAT are available on the web at http://www.satlib.org/.

URL to Code

Solvers and information on the annual satisfiability solving competition are available on the web at http://www.satlive.org/.

Recommended Reading

1. Alon, N., Kahale, N.: A spectral technique for coloring random 3-colorable graphs. SIAM J. Comput. **26**(6), 1733–1748 (1997)
2. Barthel, W., Hartmann, A.K., Leone, M., Ricci-Tersenghi, F., Weigt, M., Zecchina, R.: Hiding solutions in random satisfiability problems: A statistical mechanics approach. Phys. Rev. Lett. **88**, 188701 (2002)
3. Chen, H., Frieze, A.M.: Coloring bipartite hypergraphs. In: Cunningham, H.C., McCormick, S.T., Queyranne, M. (eds.) Integer Programming and Combinatorial Optimization, 5th International IPCO Conference, Vancouver, British Columbia, Canada, June 3–5 1996. Lecture Notes in Computer Science, vol. 1084, pp. 345–358. Springer
4. Cook, S.: The complexity of theorem-proving procedures. In: Proceedings of the 3rd Annual Symposium on Theory of Computing, pp. 151–158. Shaker Heights. May 3–5, 1971.
5. Feige, U., Mossel, E., Vilenchik, D.: Complete convergence of message passing algorithms for some satisfiability problems. In: Díaz, J., Jansen, K., Rolim, J.D.P., Zwick, U. (eds.) Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 9th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX 2006 and 10th International Workshop on Randomization and Computation, RANDOM 2006, Barcelona, Spain, August 28–30 2006. Lecture Notes in Computer Science, vol. 4110, pp. 339–350. Springer
6. Feige, U., Vilenchik, D.: A local search algorithm for 3-SAT, Tech. rep. The Weizmann Institute, Rehovat, Israel (2004)
7. Flaxman, A.D.: A spectral technique for random satisfiable 3CNF formulas. In: Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms (Baltimore, MD, 2003), pp. 357–363. ACM, New York (2003)
8. Koutsoupias, E., Papadimitriou, C.H.: On the greedy algorithm for satisfiability. Inform. Process. Lett. **43**(1), 53–55 (1992)
9. Krivelevich, M., Vilenchik, D.: Solving random satisfiable 3CNF formulas in expected polynomial time. In: SODA '06: Proceedings of the 17th annual ACM-SIAM symposium on Discrete algorith. ACM, Miami, Florida (2006)
10. Levin, L.A.: Universal enumeration problems. Probl. Pereda. Inf. **9**(3), 115–116 (1973)

Ranked Matching

2005; Abraham, Irving, Kavitha, Mehlhorn

KAVITHA TELIKEPALLI
CSA Department, Indian Institute of Science,
Bangalore, India

Keywords and Synonyms

Popular matching

Problem Definition

This problem is concerned with matching a set of *applicants* to a set of *posts*, where each applicant has a *preference list*, ranking a non-empty subset of posts in order of preference, possibly involving ties. Say that a matching M is *popular* if there is no matching M' such that the number of applicants preferring M' to M exceeds the number of applicants preferring M to M'. The ranked matching problem is to determine if the given instance admits a popular matching and if so, to compute one. There are many practical situations that give rise to such large-scale matching problems involving two sets of participants – for example, pupils and schools, doctors and hospitals – where participants of one set express preferences over the participants of the other set; an allocation determined by a popular matching can be regarded as an optimal allocation in these applications.

Notations and Definitions

An instance of the *ranked matching problem* is a bipartite graph $G = (\mathcal{A} \cup \mathcal{P}, E)$ and a partition $E = E_1 \dot\cup E_2 \ldots \dot\cup E_r$ of the edge set. Call the nodes in $\mathcal{A}$ *applicants*, the nodes in $\mathcal{P}$ *posts*, and the edges in E_i the edges of rank i. If $(a, p) \in E_i$ and $(a, p') \in E_j$ with $i < j$, say that a prefers p to p'. If $i = j$, say that a is indifferent between p and p'. An instance is *strict* if the degree of every applicant in every E_i is at most one.

A matching M is a set of edges, no two of which share an endpoint. In a matching M, a node $u \in \mathcal{A} \cup \mathcal{P}$ is either *unmatched*, or *matched* to some node, denoted by $M(u)$. Say that an applicant a *prefers* matching M' to M if (i) a is matched in M' and unmatched in M, or (ii) a is matched in both M' and M, and a prefers $M'(a)$ to $M(a)$.

Definition 1 M' is *more popular than* M, denoted by $M' \succ M$, if the number of applicants preferring M' to M exceeds the number of applicants preferring M to M'. A matching M is popular if and only if there is no matching M' that is more popular than M.

Figure 1 shows an instance with $A = \{a_1, a_2, a_3\}$, $P = \{p_1, p_2, p_3\}$, and each applicant prefers p_1 to p_2, and p_2 to p_3 (assume throughout that preferences are transitive). Consider the three symmetrical matchings $M_1 = \{(a_1, p_1), (a_2, p_2), (a_3, p_3)\}$, $M_2 = \{(a_1, p_3), (a_2, p_1), (a_3, p_2)\}$ and $M_3 = \{(a_1, p_2), (a_2, p_3), (a_3, p_1)\}$. It is easy to verify that none of these matchings is popular, since $M_1 \prec M_2$, $M_2 \prec M_3$, and $M_3 \prec M_1$. In fact, this instance admits no popular matching—the problem being, of course, that the *more popular than* relation is not acyclic, and so there need not be a maximal element.

a_1	:	p_1	p_2	p_3
a_2	:	p_1	p_2	p_3
a_3	:	p_1	p_2	p_3

Ranked Matching, Figure 1
An instance for which there is no popular matching

The *ranked matching problem* is to determine if a given instance admits a popular matching, and to find such a matching, if one exists. Popular matchings may have different sizes, and a largest such matching may be smaller than a maximum-cardinality matching. The *maximum-cardinality popular matching problem* then is to determine if a given instance admits a popular matching, and to find a *largest* such matching, if one exists.

Key Results

First consider *strict instances*, that is, instances $(\mathcal{A} \cup \mathcal{P}, E)$ where there are no ties in the preference lists of the applicants. Let n be the number of vertices and m be the number of edges in G.

Theorem 1 *For a strict instance $G = (\mathcal{A} \cup \mathcal{P}, E)$, it is possible to determine in $O(m + n)$ time if G admits a popular matching and compute one, if it exists.*

Theorem 2 *Find a maximum-cardinality popular matching of a strict instance $G = (\mathcal{A} \cup \mathcal{P}, E)$, or determine that no such matching exists, in $O(m + n)$ time.*

Next consider the general problem, where preference lists may have ties.

Theorem 3 *Find a popular matching of $G = (\mathcal{A} \cup \mathcal{P}, E)$, or determine that no such matching exists, in $O(\sqrt{n}m)$ time.*

Theorem 4 *Find a maximum-cardinality popular matching of $G = (\mathcal{A} \cup \mathcal{P}, E)$, or determine that no such matching exists, in $O(\sqrt{n}m)$ time.*

Techniques

Our results are based on a novel characterization of popular matchings. For exposition purposes, create a unique *last resort* post $l(a)$ for each applicant a and assign the edge $(a, l(a))$ a rank higher than any edge incident on a. In this way, assume that every applicant is matched, since any unmatched applicant can be allocated to his/her last resort. From now on then, matchings are *applicant-complete*, and the size of a matching is just the number of applicants not matched to their last resort. Also assume that instances have no gaps, i. e., if an applicant has a rank i edge incident to it then it has edges of all smaller ranks incident to it. First outline the characterization in strict instances and then extend it to general instances.

Strict Instances For each applicant a, let $f(a)$ denote the most preferred post on a's preference list. That is, $(a, f(a)) \in E_1$. Call any such post p an *f-post*, and denote by $f(p)$ the set of applicants a for which $f(a) = p$.

For each applicant a, let $s(a)$ denote the most preferred non-f-post on a's preference list; note that $s(a)$ must exist, due to the introduction of $l(a)$. Call any such post p an *s-post*, and remark that f-posts are disjoint from s-posts.

Using the definitions of f-posts and s-posts, show three conditions that a popular matching must satisfy.

Lemma 5 *Let M be a popular matching.*

1. *For every f-post p, (i) p is matched in M, and (ii) $M(p) \in f(p)$.*
2. *For every applicant a, $M(a)$ can never be strictly between $f(a)$ and $s(a)$ on a's preference list.*
3. *For every applicant a, $M(a)$ is never worse than $s(a)$ on a's preference list.*

It is then shown that these three necessary conditions are also sufficient. This forms the basis of the following preliminary characterization of popular matchings.

Lemma 6 *A matching M is popular if and only if (i) every f-post is matched in M, and (ii) for each applicant a, $M(a) \in \{f(a), s(a)\}$.*

Given an instance graph $G = (\mathcal{A} \cup \mathcal{P}, E)$, define the *reduced graph* $G' = (\mathcal{A} \cup \mathcal{P}, E')$ as the subgraph of G containing two edges for each applicant a: one to $f(a)$, the other to $s(a)$. The authors remark that G' need not admit an applicant-complete matching, since $l(a)$ is now isolated whenever $s(a) \neq l(a)$. Lemma 6 shows that a matching is popular if and only if it belongs to the graph G' and it matches every f-post. Recall that all popular matchings are applicant-complete through the introduction of last resorts. Hence, the following characterization is immediate.

Theorem 7 *M is a popular matching of G if and only if (i) every f-post is matched in M, and (ii) M is an applicant-complete matching of the reduced graph G'.*

The characterization in Theorem 7 immediately suggests the following algorithm for solving the popular matching problem. Construct the reduced graph G'. If G' does not admit an applicant-complete matching, then G admits no popular matching. If G' admits an applicant-complete matching M, then modify M so that every f-post is matched. So for each f-post p that is unmatched in M, let a be any applicant in $f(p)$; remove the edge $(a, M(a))$ from M and instead match a to p. This algorithm can be implemented in $O(m + n)$ time. This shows Theorem 1.

Now, consider the maximum-cardinality popular matching problem. Let $\mathcal{A}_1$ be the set of all applicants a with $s(a) = l(a)$. Let $\mathcal{A}_1$ be the set of all applicants with $s(a) = l(a)$. Our target matching must satisfy conditions (i) and (ii) of Theorem 7, and among all such matchings, allocate the fewest $\mathcal{A}_1$-applicants to their last resort. This scheme can be implemented in $O(m + n)$ time. This proves Theorem 2.

General Instances For each applicant a, let $f(a)$ denote the *set* of first-ranked posts on a's preference list. Again, refer to all such posts p as *f-posts*, and denote by $f(p)$ the set of applicants a for which $p \in f(a)$. It may no longer be possible to match *every* f-post p with an applicant in $f(p)$ (as in Lemma 5), since, for example, there may now be more f-posts than applicants. Let M be a popular matching of some instance graph $G = (\mathcal{A} \cup \mathcal{P}, E)$. Define the *first-choice graph* of G as $G_1 = (\mathcal{A} \cup \mathcal{P}, E_1)$, where E_1 is the set of all rank one edges. Next the authors show the following lemma.

Lemma 8 *Let M be a popular matching. Then $M \cap E_1$ is a maximum matching of G_1.*

Next, work towards a generalized definition of $s(a)$. Restrict attention to rank-one edges, that is, to the graph G_1 and using M_1, partition $\mathcal{A} \cup \mathcal{P}$ into three disjoint sets. A node v is *even* (respectively *odd*) if there is an even (respectively odd) length alternating path (with respect to M_1) from an unmatched node to v. Similarly, a node v is *unreachable* if there is no alternating path (w.r.t. M_1) from an unmatched node to v. Denote by $\mathcal{E}$, $\mathcal{O}$, and $\mathcal{U}$ the sets of even, odd, and unreachable nodes, respectively. Conclude the following facts about $\mathcal{E}$, $\mathcal{O}$, and $\mathcal{U}$ by using the well-known Gallai–Edmonds decomposition theorem.

- **(a)** $\mathcal{E}$, $\mathcal{O}$, and $\mathcal{U}$ are pairwise disjoint. Every maximum matching in G_1 partitions the vertex set into the same partition of even, odd, and unreachable nodes.
- **(b)** In any maximum-cardinality matching of G_1, every node in $\mathcal{O}$ is matched with some node in $\mathcal{E}$, and every node in $\mathcal{U}$ is matched with another node in $\mathcal{U}$. The size of a maximum-cardinality matching is $|\mathcal{O}| + |\mathcal{U}|/2$.
- **(c)** No maximum-cardinality matching of G_1 contains an edge between two nodes in $\mathcal{O}$, or a node in $\mathcal{O}$ and

a node in $\mathcal{U}$. And there is no edge in G_1 connecting a node in $\mathcal{E}$ with a node in $\mathcal{U}$.

The above facts motivate the following definition of $s(a)$: let $s(a)$ be the set of most preferred posts in a's preference list that are *even* in G_1 (note that $s(a) \neq \emptyset$, since $l(a)$ is always even in G_1). Recall that our original definition of $s(a)$ led to parts (2) and (3) of Lemma 5 which restrict the set of posts to which an applicant can be matched in a popular matching. This shows that the generalized definition leads to analogous results here.

Lemma 9 *Let M be a popular matching. Then for every applicant a, $M(a)$ can never be strictly between $f(a)$ and $s(a)$ on a's preference list and $M(a)$ can never be worse than $s(a)$ in a's preference list.*

The following characterization of popular matchings is formed.

Lemma 10 *A matching M is popular in G if and only if (i) $M \cap E_1$ is a maximum matching of G_1, and (ii) for each applicant a, $M(a) \in f(a) \cup s(a)$.*

Given an instance graph $G = (\mathcal{A} \cup \mathcal{P}, E)$, we define the *reduced graph* $G' = (\mathcal{A} \cup \mathcal{P}, E')$ as the subgraph of G containing edges from each applicant a to posts in $f(a) \cup s(a)$. The authors remark that G' need not admit an applicant-complete matching, since $l(a)$ is now isolated whenever $s(a) \neq \{l(a)\}$. Lemma 11 tells us that a matching is popular if and only if it belongs to the graph G' and it is a maximum matching on rank one edges. Recall that all popular matchings are applicant-complete through the introduction of last resorts. Hence, the following characterization is immediate.

Theorem 11 *M is a popular matching of G if and only if (i) $M \cap E_1$ is a maximum matching of G_1, and (ii) M is an applicant-complete matching of G'.*

Using the characterization in Theorem 11, the authors now present an efficient algorithm for solving the ranked matching problem.

Popular-Matching($G = (\mathcal{A} \cup \mathcal{P}, E)$)

1. Construct the graph $G' = (\mathcal{A} \cup \mathcal{P}, E')$, where $E' = \{(a, p) \mid p \in f(a) \cup s(a),\ a \in \mathcal{A}\}$.
2. Compute a maximum matching M_1 on rank one edges i. e., M_1 is a maximum matching in $G_1 = (\mathcal{A} \cup \mathcal{P}, E_1)$. ($M_1$ is also a matching in G' because $E' \supseteq E_1$)
3. Delete all edges in G' connecting two nodes in the set $\mathcal{O}$ or a node in $\mathcal{O}$ with a node in $\mathcal{U}$, where $\mathcal{O}$ and $\mathcal{U}$ are the sets of odd and unreachable nodes of $G_1 = (\mathcal{A} \cup \mathcal{P}, E_1)$.
   Determine a maximum matching M in the modified graph G' by augmenting M_1.
4. If M is not applicant-complete, then declare that there is no popular matching in G.
   Else return M.

The matching returned by the algorithm Popular-Matching is an applicant-complete matching in G' and it is a maximum matching on rank one edges. So the correctness of the algorithm follows from Theorem 11. It is easy to see that the running time of this algorithm is $O(\sqrt{n}m)$. The algorithm of Hopcroft and Karp [7] is uesd to compute a maximum matching in G_1 and identify the set of edges E' and construct G' in $O(\sqrt{n}m)$ time. Repeatedly augment M_1 (by the Hopcroft–Karp algorithm) to obtain M. This proves Theorem 3.

It is now a simple matter to solve the maximum-cardinality popular matching problem. Assume that the instance $G = (\mathcal{A} \cup \mathcal{P}, E)$ admits a popular matching. (Otherwise, the process is done.) In order to compute an applicant-complete matching in G' that is a maximum matching on rank one edges and which maximizes the number of applicants not matched to their last resort, first compute an arbitrary popular matching M' and remove all edges of the form $(a, l(a))$ from M' and from the graph G'. Call the resulting subgraph of G' as H. Determine a maximum matching N in H by augmenting M'. N need not be a popular matching, since it need not be a maximum matching in the graph G'. However, this is easy to mend. Determine a maximum matching M in G' by augmenting N. It is easy to show that M is a popular matching which maximizes the number of applicants not matched to their last resort. Since the algorithm takes $O(\sqrt{n}m)$ time, Theorem 4 is shown.

Applications

The bipartite matching problem with a graded edge set is well-studied in the economics literature, see for example [1,10,12]. It models some important real-world problems, including the allocation of graduates to training positions [8], and families to government-owned housing [11]. The concept of a popular matching was first introduced by Gardenfors [5] under the name *majority assignment* in the context of the stable marriage problem [4,6].

Various other definitions of optimality have been considered. For example, a matching is *Pareto-optimal* [1,2,10] if no applicant can improve his/her allocation (say by exchanging posts with another applicant) without requiring some other applicant to be worse off. Stronger definitions exist: a matching is *rank-maximal* [9] if it allocates the maximum number of applicants to their first choice, and then subject to this, the maximum number to their second choice, and so on. A matching is *maximum utility* if it maximizes $\sum_{(a,p)\in M} u_{a,p}$, where $u_{a,p}$ is the utility of allocating post p to applicant a. Neither rank-maximal nor maximum-utility matchings are necessarily popular.

Cross References

► Hospitals/Residents Problem
► Maximum Matching
► Weighted Popular Matchings

Recommended Reading

1. Abdulkadiroĝlu, A., Sönmez, T.: Random serial dictatorship and the core from random endowments in house allocation problems. Econom. **66**(3), 689–701 (1998)
2. Abraham, D.J., Cechlárová, K., Manlove, D.F., Mehlhorn, K.: Pareto-optimality in house al- location problems. In: Proceedings of the 15th International Symposium on Algorithms and Computation, (LNCS, vol. 3341), pp. 3–15. Springer, Sanya, Hainan (2004)
3. Abraham, D.J., Irving, R.W., Kavitha, T., Mehlhorn, K.: Popular matchings. In: Proceedings of the 16th ACM-SIAM Symposium on Discrete Algorithms, pp. 424–432. SIAM, Vancouver (2005)
4. Gale, D., Shapley, L.S.: College admissions and the stability of marriage. Am. Math. Mon. **69**, 9–15 (1962)
5. Gardenfors, P.: Match Making: assignments based on bilateral preferences. Behav. Sci. **20**, 166–173 (1975)
6. Guseld, D., Irving, R.W.: The Stable Marriage Problem: Structure and Algorithms. MIT Press, Cambridge (1989)
7. Hopcroft, J.E., Karp, R.M.: A $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs. SIAM J. Comput. **2**, 225–231 (1973)
8. Hylland, A., Zeckhauser, R.: The ecient allocation of individuals to positions. J. Political Econ. **87**(2), 293–314 (1979)
9. Irving, R.W., Kavitha, T., Mehlhorn, K., Michail, D., Paluch, K.: Rank-maximal matchings. In: Proceedings of the 15th ACM-SIAM Symposium on Discrete Algorithms, pp. 68–75. SIAM, New Orleans (2004)
10. Roth, A.E., Postlewaite, A.: Weak versus strong domination in a market with indivisible goods. J. Math. Econ. **4**, 131–137 (1977)
11. Yuan, Y.: Residence exchange wanted: a stable residence exchange problem. Eur. J. Oper. Res. **90**, 536–546 (1996)
12. Zhou, L.: On a conjecture by Gale about one-sided matching problems. J. Econ. Theory **52**(1), 123–135 (1990)

Rank and Select Operations on Binary Strings

1974; Elias

Naila Rahman, Rajeev Raman
Department of Computer Science,
University of Leicester, Leicester, UK

Keywords and Synonyms

Binary bit-vector; Compressed bit-vector, Rank and select dictionary; Fully indexable dictionary (FID)

Problem Definition

Given a static sequence $\boldsymbol{b} = b_1 \dots b_m$ of m bits, to preprocess the sequence and to create a space-efficient data structure that supports the following operations rapidly:

$\mathsf{rank}_1(i)$ takes an index i as input, $1 \le i \le m$, and returns the number of **1**s among $b_1 \dots b_i$.

$\mathsf{select}_1(i)$ takes an index $i \ge 1$ as input, and returns the position of the ith **1** in $\boldsymbol{b}$, and -1 if i is greater than the number of **1**s in $\boldsymbol{b}$.

The operations rank_0 and select_0 are defined analogously for the **0**s in $\boldsymbol{b}$. As $\mathsf{rank}_0(i) = i - \mathsf{rank}_1(i)$, one considers just rank_1 (abbreviated to rank), and refers to select_0 and select_1 collectively as select. In what follows, $|\boldsymbol{x}|$ denotes the length of a bit sequence $\boldsymbol{x}$ and $w(\boldsymbol{x})$ denotes the number of **1**s in it. $\boldsymbol{b}$ is always used to denote the input bit sequence, m to denote $|\boldsymbol{b}|$ and n to denote $w(\boldsymbol{b})$.

Models of Computation, Time and Space Bounds

Two models of computation are commonly considered. One is the *unit-cost RAM* model with word size $O(\lg m)$ bits [1]. The other model, which is particularly useful for proving lower bounds, is the *bit-probe* model, where the data structure is stored in bit-addressable memory, and the complexity of answering a query is the worst-case number of bits of the data structure that are probed by the algorithm to answer that query. In the RAM model, the algorithm can read $O(\lg m)$ *consecutive* bits in one step, so supporting all operations in $O(1)$ time on the RAM model implies a solution that uses $O(\lg m)$ bit-probes, but the converse is not true.

This entry considers three variants of the problem: in each variant, rank and select must be supported in $O(1)$ time on the RAM model, or in $O(\lg m)$ bit-probes. However, the use of memory varies:

Problem 1 (Bit-Vector) *The overall space used must be $m + o(m)$ bits.*

Problem 2 (Bit-Vector Index) *$\boldsymbol{b}$ is given in read-only memory and the algorithm can create auxiliary data structures (called indices) which must use $o(m)$ bits.*

Indices allow the representation of $\boldsymbol{b}$ to be de-coupled from the auxiliary data structure, e. g., $\boldsymbol{b}$ can be stored (in a potentially highly compressed form) in a data structure such as that of [6,9,17] which allows access to $O(\lg m)$ consecutive bits of $\boldsymbol{b}$ in $O(1)$ time on the RAM model. Most bit-vectors developed to date are bit-vector indices.

Recalling that $n = w(\boldsymbol{b})$, observe that if m and n are known to an algorithm, there are only $l = \binom{m}{n}$ possibilities for $\boldsymbol{b}$, so an information-theoretically optimal encoding of $\boldsymbol{b}$ would require $B(m, n) = \lceil \lg l \rceil$ bits (it can be verified that $B(m, n) < m$ for all m, n). The next problem is:

Problem 3 (Compressed Bit-Vector) *The overall space used must be $B(m, n) + o(n)$ bits.*

It is helpful to understand the asymptotics of $B(m, n)$ in order to appreciate the difference between the bit-vector and the compressed bit-vector problems:

- Using standard approximations of the factorial function, one can show [14] that:
$$B(m, n) = n \lg(m/n) + n \lg e + O(n^2/m) \quad (1)$$
If $n = o(m)$, then $B(m, n) = o(m)$, and if such a sparse sequence $\boldsymbol{b}$ were represented as a compressed bit-vector, then it would occupy $o(m)$ bits, rather than $m + o(m)$ bits.
- $B(m, n) = m - O(\lg m)$, whenever $|m/2 - n| = O(\sqrt{m \lg m})$. In such cases, a compressed bit-vector will take about the same amount of space as a bit-vector.
- Taking $p = n/m$, $H_0(\boldsymbol{b}) = (1/p) \lg(1/p) + (1/(1-p)) \lg(1/(1-p))$ is the *empirical zeroth-order entropy* of $\boldsymbol{b}$. If $\boldsymbol{b}$ is compressed using an 'entropy' compressor such as non-adaptive arithmetic coding [18], the size of the compressed output is at least $mH_0(\boldsymbol{b})$ bits. However, $B(m, n) = mH_0(\boldsymbol{b}) - O(\log m)$. Applying Golomb coding to the 'gaps' between successive **1**s, which is the best way to compress bit sequences that represent inverted lists [18], also gives a space usage close to $B(m, n)$ [4].

Related Problems

Viewing $\boldsymbol{b}$ as the characteristic vector of a set $S \subseteq U = \{1, \ldots, m\}$, note that the well-known *predecessor* problem – given $y \in U$, return $pred(y) = \max\{z \in S | z \leq y\}$ – may be implemented as $\mathsf{select}_1(\mathsf{rank}_1(y))$. One may also view $\boldsymbol{b}$ as a *multiset* of size $m - n$ over the universe $\{1, \ldots, n+1\}$ [5]. First, append a **1** to $\boldsymbol{b}$. Then, take each of the $n+1$ **1**s to be the elements of the universe, and the number of consecutive **0**s immediately preceding a **1** to indicate their multiplicities. For example, $\boldsymbol{b}$ = **01100100** maps to the multiset $\{1, 3, 3, 4, 4\}$. Seen this way, $\mathsf{select}_1(i) - i$ on $\boldsymbol{b}$ gives the number of items in the multiset that are $\leq i$, and $\mathsf{select}_0(i) - i + 1$ gives the value of the ith element of the multiset.

Lower-Order Terms

From an asymptotic viewpoint, the space utilization is dominated by the main terms in the space bound. However, the second (apparently lower-order) terms are of interest for several reasons, primarily because the lower-order terms are extremely significant in determining practical space usage, and also because non-trivial space bounds have been proven for the size of the lower-order terms.

Key Results

Reductions

It has been already noted that rank_0 and rank_1 reduce to each other, and that operations on multisets reduce to select operations on a bit sequence. Some other reductions, whereby one can support operations on $\boldsymbol{b}$ by performing operations on bit sequences derived from $\boldsymbol{b}$ are:

Theorem 1

(a) rank *reduces to* select_0 *on a bit sequence* $\boldsymbol{c}$ *such that* $|\boldsymbol{c}| = m + n$ *and* $w(\boldsymbol{c}) = n$.

(b) *If* $\boldsymbol{b}$ *has no consecutive* **1***s, then* select_0 *on* $\boldsymbol{b}$ *can be reduced to* rank *on a bit sequence* $\boldsymbol{c}$ *such that* $|\boldsymbol{c}| = m - n$ *and* $w(\boldsymbol{c})$ *is either* $n - 1$ *or* n.

(c) *From* $\boldsymbol{b}$ *one can derive two bit sequences* $\boldsymbol{b_0}$ *and* $\boldsymbol{b_1}$ *such that* $|\boldsymbol{b_0}| = m - n$, $|\boldsymbol{b_1}| = n$, $w(\boldsymbol{b_0}), w(\boldsymbol{b_1}) \leq \min\{m - n, n\}$ *and* select_0 *and* select_1 *on* $\boldsymbol{b}$ *can be supported by supporting* select_1 *and* rank *on* b_0 *and* b_1.

Parts (a) and (b) follow from Elias's observations on multiset representations (see the "Related Problems" paragraph), specialized to sets. For part (a), create $\boldsymbol{c}$ from $\boldsymbol{b}$ by adding a **0** after every **1**. For example, if $\boldsymbol{b}$ = **01100100** then $\boldsymbol{c}$ = **01010001000**. Then, $\mathsf{rank}_1(i)$ on $\boldsymbol{b}$ equals $\mathsf{select}_0(i) - i$ on $\boldsymbol{c}$. For part (b), essentially invert the mapping of part (a). Part (c) is shown in [3].

Bit-Vector Indices

Theorem 2 ([8]) *There is an index of size* $(1 + o(1))(m \lg\lg m / \lg m) + O(m/\lg m)$ *that supports* rank *and* select *in O(1) time on the RAM model.*

Elias previously gave an $o(m)$-bit index that supported select in $O(\lg m)$ bit-probes on average (where the average was computed across all select queries). Jacobson gave $o(m)$-bit indices that supported rank and select in $O(\lg m)$ bit-probes in the worst case. Clark and Munro [2] gave the first $o(m)$-bit indices that support both rank and select in $O(1)$ time on the RAM. A matching lower bound on the size of indices has also been shown (this also applies to indices which support rank and select in $O(1)$ time on the RAM model):

Theorem 3 ([8]) *Any index that allows* rank *or* select$_1$ *to be supported in* $O(\lg m)$ *bit-probes has size* $\Omega(m \lg\lg m / \lg m)$ *bits.*

Compressed Bit-Vectors

Theorem 4 *There is a compressed bit-vector that uses:*

(a) $B(m, n) + O(m \lg\lg m / \lg m)$ *bits and supports rank and select in O(1) time.*
(b) $B(m, n) + O(n(\lg\lg n)^2 / \lg n)$ *bits and supports rank in O(1) time, when* $n = m/(\lg m)^{O(1)}$.
(c) $B(m, n) + O(n \lg\lg n / \sqrt{\lg n})$ *bits and supports select$_1$ in O(1) time.*

Theorem 4(a) and (c) were shown by Raman et al. [16] and Theorem 4(b) by Pagh [14]. Note that Theorem 4(a) has a lower-order term that is $o(m)$, rather than $o(n)$ as required by the problem statement. As compressed bit-vectors must represent $\boldsymbol{b}$ compactly, they are not bit-vector indices, and the lower bound of Theorem 3 does not apply to compressed bit-vectors. Coarser lower bounds are obtained by reduction to the predecessor problem on sets of integers, for which tight upper and lower bounds in the RAM model are now known. In particular the work of [15] implies:

Theorem 5 *Let* $U = \{1, \ldots, M\}$ *and let* $S \subseteq U, |S| = N$. *Any data structure on a RAM with word size* $O(\lg M)$ *bits that occupies at most* $O(N \lg M)$ *bits of space can support predecessor queries on S in O(1) time only when* $N = M/(\lg M)^{O(1)}$ *or* $N = (\lg M)^{O(1)}$.

As noted in the paragraph "Related Problems", the predecessor problem can be solved by the use of rank and select$_1$ operations. Thus, Theorem 5 has consequences for compressed bit-vector data structures, which are spelt out below:

Corollary 1 *There is no data structure that uses* $B(m, n) + o(n)$ *bits and supports either* rank *or* select$_0$ *in O(1) time unless* $n = m/(\lg m)^{O(1)}$, *or* $n = (\log m)^{O(1)}$.

Given a set $S \subseteq U = \{1, \ldots, m\}, |S| = n$, we have already noted that the predecessor problem on S is equivalent to rank and select$_1$ on a bit-vector $\boldsymbol{c}$ with $w(\boldsymbol{c}) = n$, and $|\boldsymbol{c}| = m$. However, $B(m, n) + o(n) = O(n \lg m)$. Thus, given a bit-vector that uses $B(m, n) + o(n)$ bits and supports rank in $O(1)$ time for $m = n(\lg n)^{\omega(1)}$, we can augment it with the trivial $O(1)$-time data structure for select$_1$, that stores the value of select$_1$ (i) for $i = 1, \ldots, n$ (which occupies a further $O(n \lg m)$ bits), solving the predecessor problem in $O(1)$ time, a contradiction. The hardness of select$_0$ is shown in [16], but follows easily from Theorem 1(a) and Eq. (1).

Applications

There are a vast number of applications of bit-vectors in succinct and compressed data structures (see e. g. [12]). Such data structures are used for, e. g., text indexing, compact representations of graphs and trees, and representations of semi-structured (XML) data.

Experimental Results

Several teams have implemented bit-vectors and compressed bit-vectors. When implementing bit-vectors for good practical performance, both in terms of speed and space usage, the lower-order terms are very important, even for uncompressed bit-vectors[1], and can dominate the space usage even for bit-vector sizes that are at the limit of conceivable future practical interest. Unfortunately, this problem may not be best addressed purely by a theoretical analysis of the lower-order terms. Bit-vectors work by partitioning the input bit sequence into (usually equal-sized) blocks at several levels of granularity – usually 2–3 levels are needed to obtain a space bound of $m + o(m)$ bits. However, better space usage – as well as better speed – in practice can be obtained by reducing the number of levels, resulting in space bounds of the form $(1 + \epsilon)m$ bits, for any $\epsilon > 0$, with support for rank and select in $O(1/\epsilon)$ time.

Clark [2] implemented bit-vectors for external-memory suffix trees. More recently, an implementation using ideas of Clark and Jacobson was used by [7], which occupied $(1+\epsilon)m$ bits and supported operations in $O(1/\epsilon)$ time. Using a substantially different approach, Kim et al. [11] gave a bit-vector that takes $(2 + \epsilon)n$ bits to support rank and select. Experiments using bit sequences derived from real-world data in [3,4] showed that if parameters are set to ensure that [11] and [7] use similar space – on *typical* inputs – the Clark–Jacobson implementation of [7] is somewhat faster than an implementation of [11]. On some inputs, the Clark–Jacobson implementation can use significantly more space, whereas Kim et al.'s bit-vector appears to have stable space usage; Kim et al.'s bit-vector may also be superior for somewhat sparse bit-vectors. Combining ideas from [7,11], a third practical bit-vector (which is not

[1]For compressed bit-vectors, the 'lower-order' $o(m)$ or $o(n)$ term can dominate $B(m, n)$, but this is not our concern here.

a bit-vector index) was described in [4], and appears to have desirable features of both [11] and [7]. A first implementational study on compressed bit-vectors can be found in [13] (compressed bit-vectors supporting only select_1 were considered in [4]).

URL to Code

Bit-vector implementations from [3,4,7] can be found at http://hdl.handle.net/2381/318.

Cross References

► Arithmetic Coding for Data Compression
► Compressed Text Indexing
► Succinct Encoding of Permutations: Applications to Text Indexing
► Tree Compression and Indexing

Recommended Reading

1. Aho, A.V., Hopcroft, J.E., Ullman, J.D.: The Design and Analysis of Computer Algorithms. Addison-Wesley (1974)
2. Clark, D., Munro, J.I.: Efficient suffix trees on secondary storage. In: Proc. 7th ACM-SIAM SODA, pp. 383–391 (1996)
3. Delpratt, O., Rahman, N., Raman, R.: Engineering the LOUDS succinct tree representation. In: Proc. WEA 2006. LNCS, vol. 4007, pp. 134–145. Springer, Berlin (2006)
4. Delpratt, O., Rahman, N., Raman, R.: Compressed prefix sums. In: Proc. SOFSEM 2007. LNCS, vol. 4362, pp. 235–247 (2007)
5. Elias, P.: Efficient storage retrieval by content and address of static files. J. ACM, **21**(2):246–260 (1974)
6. Ferragina, P., Venturini, R.: A simple storage scheme for strings achieving entropy bounds. Theor. Comput. Sci. **372**, 115–121 (2007)
7. Geary, R.F., Rahman, N., Raman, R., Raman, V.: A simple optimal representation for balanced parentheses. Theor. Comput. Sci. **368**, 231–246 (2006)
8. Golynski, A.: Optimal lower bounds for rank and select indexes. In: Proc. ICALP 2006, Part I. LNCS, vol. 4051, pp. 370–381 (2006)
9. González, R., Navarro, G.: Statistical encoding of succinct data structures. In: Proc. CPM 2006. LNCS, vol. 4009, pp. 294–305. Springer, Berlin (2006)
10. Jacobson, G.: Space-efficient static trees and graphs. In: Proc. 30th FOCS, pp. 549–554 (1989)
11. Kim, D.K., Na, J.C., Kim, J.E., Park, K.: Efficient implementation of Rank and Select functions for succinct representation. In: Proc. WEA 2005. LNCS, vol. 3505, pp. 315–327 (2005)
12. Munro, J.I., Srinivasa Rao, S.: Succinct representation of data structures. In: Mehta, D., Sahni, S. (eds.) Handbook of Data Structures with Applications, Chap 37. Chapman and Hall/CRC Press (2005)
13. Okanohara, D., Sadakane, K.: Practical entropy-compressed rank/select dictionary. In: Proc. 9th ACM-SIAM Workshop on Algorithm Engineering and Experiments (ALENEX '07), SIAM, to appear (2007)
14. Pagh, R.: Low redundancy in static dictionaries with constant query time. SIAM J. Comput. **31**, 353–363 (2001)
15. Patrascu, M., Thorup, M.: Time-space trade-offs for predecessor search. In: Proc. 38th ACM STOC, pp. 232–240 (2006)
16. Raman, R., Raman, V., Rao, S.S.: Succinct indexable dictionaries, with applications to representing *k*-ary trees and multisets. In: Proc. 13th ACM-SIAM SODA, pp. 233–242 (2002)
17. Sadakane, K., Grossi, R.: Squeezing succinct data structures into entropy bounds. In: Proc. 17th ACM-SIAM SODA, pp. 1230–1239. ACM Press (2006)
18. Witten, I., Moffat, A., Bell, I.: Managing Gigabytes, 2nd edn. Morgan Kaufmann (1999)

Rate Adjustment and Allocation

► Schedulers for Optimistic Rate Based Flow Control

Rate-Monotonic Scheduling

1973; Liu, Layland

NATHAN FISHER, SANJOY BARUAH
Department of Computer Science,
University of North Carolina, Chapel Hill, NC, USA

Keywords and Synonyms

Real-time systems; Static-priority scheduling; Fixed-priority scheduling; Rate-monotonic analysis

Problem Definition

Liu and Layland [9] introduced rate-monotonic scheduling in the context of the scheduling of recurrent real-time processes upon a computing platform comprised of a single preemptive processor.

The Periodic Task Model

The *periodic task* abstraction models real-time processes that make repeated requests for computation. As defined in [9], each periodic task τ_i is characterized by an ordered pair of positive real-valued parameters (C_i, T_i), where C_i is the *worst-case execution requirement* and T_i the *period* of the task. The requests for computation that are made by task τ_i (subsequently referred to as *jobs* that are *generated* by τ_i) satisfy the following assumptions:

A1: τ_i's first job arrives at system start time (assumed to equal time zero), and subsequent jobs arrive every T_i time units. I.e., one job arrives at time-instant $k \times T_i$ for all integer $k \geq 0$.

A2: Each job needs to execute for at most C_i time units. I.e., C_i is the maximum amount of time that a processor would require to execute each job of τ_i, without interruption.

A3: Each job of τ_i must complete before the next job arrives. That is, each job of task τ_i must complete execution by a *deadline* that is T_i time-units after its arrival time.

A4: Each task is *independent* of all other tasks – the execution of any job of task τ_i is not contingent on the arrival or completion of jobs of any other task τ_j.

A5: A job of τ_i may be *preempted* on the processor without additional execution cost. In other words, if a job of τ_i is currently executing, then it is permitted that this execution be halted, and a job of a different task τ_j begin execution immediately.

A periodic task system $\tau \overset{\text{def}}{=} \{\tau_1, \tau_2, \ldots, \tau_n\}$ is a collection of n periodic tasks. The *utilization* $U(\tau)$ is defined as follows:

$$U(\tau) \overset{\text{def}}{=} \sum_{i=1}^{n} C_i/T_i \; . \tag{1}$$

Intuitively, this denotes the fraction of time that may be spent by the processor executing jobs of tasks in τ, in the worst case.

The Rate-Monotonic Scheduling Algorithm

A (uniprocessor) scheduling algorithm determines which task executes on the shared processor at each time-instant. If a scheduling algorithm is guaranteed to always meet all deadlines when scheduling a task system τ, then τ is said to be *schedulable* with respect to that scheduling algorithm.

Many scheduling algorithms work as follows: At each time-instant, they assign a priority to each job, and select for execution the greatest-priority job with remaining execution. A *static priority* (often called *fixed priority*) scheduling algorithm for scheduling periodic tasks is one in which it is required that all the jobs of each periodic task be assigned the same priority.

Liu and Layland [9] proposed the *rate-monotonic* (RM) static priority scheduling algorithm, which assigns priority to jobs according to the period parameter of the task that generates them: *the smaller the period, the higher the priority*. Hence if $T_i < T_j$ for two tasks τ_i and τ_j, then each job of τ_i has higher priority than all jobs of τ_j and hence any executing job of τ_j will be preempted by the arrival of one of τ_i's jobs. Ties may be broken arbitrarily but consistently – if $T_i = T_j$, then either all jobs of τ_i are assigned higher priority than all jobs of τ_j, or all jobs of τ_j are assigned higher priority than all jobs of τ_i.

Key Results

Results from the original paper by Liu and Layland [9] are presented in Sect. "Results from [9]" below; results extending the original work are briefly described in Sect. "Results since [9]".

Results from [9]

Optimality Liu and Layland were concerned with designing "good" static priority scheduling algorithms. They defined a notion of optimality for such algorithms: A static priority algorithm $\mathcal{A}$ is *optimal* if any periodic task system that is schedulable with respect to some static priority algorithm is also schedulable with respect to $\mathcal{A}$.

Liu and Layland obtained the following result for the rate-monotonic scheduling algorithm (RM):

Theorem 1 *For periodic task systems,* RM *is an optimal static priority scheduling algorithm.*

Schedulability Testing A *schedulability test* for a particular scheduling algorithm determines, for any periodic task system τ, whether τ is schedulable with respect to that scheduling algorithm. A schedulability test is said to be *exact* if it is the case that it correctly identifies all schedulable task systems, and *sufficient* if it identifies some, but not necessarily all, schedulable task systems.

In order to derive good schedulability tests for the rate-monotonic scheduling algorithm, Liu and Layland considered the concept of *response time*. The response time of a job is defined as the elapsed time between the arrival of a job and its completion time in a schedule; the response time of a task is defined to be the largest response time that may be experienced by one its jobs. For static priority scheduling, Liu and Layland obtained the following result on the response time:

Theorem 2 *The maximum response time for a periodic task τ_i occurs when a job of τ_i arrives simultaneously with jobs of all higher-priority tasks. Such a time-instant is known as the critical instant for task τ_i.*

Observe that the critical instant of the lowest-priority task in a periodic task system is also a critical instant for all tasks of higher priority. An immediate consequence of the previous theorem is that the response-time of each task in the periodic task system can be obtained by simulating the scheduling of the periodic task system starting at the critical instant of the lowest-priority task. If the response time for each task τ_i obtained from such simulation does not exceed T_i, then the task system will always meet all deadlines when scheduled according to the given priority assignment. This argument immediately gives rise to a schedulability analysis test [7] for any static priority scheduling algorithm. Since the simulation may need to be carried out until $\max_{i=1}^{n}\{T_i\}$, this schedulability test has run-time pseudo-polynomial in the representation of the task system:

Theorem 3 ([7]) *Exact rate-monotonic schedulability testing of a periodic task system may be done in time pseudo-polynomial in the representation in the task system.*

Liu and Layland also derived a polynomial-time sufficient (albeit not exact) schedulability test for RM, based upon the utilization of the task system:

Theorem 4 *Let n denote the number of tasks in periodic task system τ. If $U(\tau) \leq n(2^{1/n} - 1)$, then τ is schedulable with respect to the RM scheduling algorithm.*

Results since [9]

The utilization-bound sufficient schedulability test (Theorem 4) was shown to be tight in the sense that for all n, there are unschedulable task systems comprised of n tasks with utilization exceeding $n(2^{1/n} - 1)$ by an arbitrarily small amount. However, tests have been devised that exploit more knowledge about tasks' period parameters. For instance, Kuo and Mok [6] provide a potentially superior utilization bound for task systems in which the task period parameters tend to be harmonically related – exact multiples of one another. Suppose that a collection of numbers is said to comprise a *harmonic chain* if for every two numbers in the set, it is the case that one is an exact multiple of the other. Let $\tilde{n}$ denote the minimum number of harmonic chains into which the period parameters $\{T_i\}_{i=1}^{n}$ of tasks in τ may be partitioned; a sufficient condition for task system τ to be RM-schedulable is that

$$U(\tau) \leq \tilde{n}(2^{1/\tilde{n}} - 1)\,.$$

Since $\tilde{n} \leq n$ for all task systems τ, this utilization bound above is never inferior to the one in Theorem 4, and is superior for all τ for which $\tilde{n} < n$.

A different polynomial-time schedulability test was proposed by Bini, Buttazzo, and Buttazzo [3]: they showed that

$$\prod_{i=1}^{n}((C_i/T_i) + 1) \leq 2$$

is sufficient to guarantee that the periodic task system $\{\tau_1, \tau_2, \ldots, \tau_n\}$ is rate-monotonic schedulable. This test is commonly referred to as the *hyperbolic* schedulability test for rate-monotonic schedulability. The hyperbolic test is in general known to be superior to the utilization-based test of Theorem 4 – see [3] for details.

Other work done since the seminal paper of Liu and Layland has focused on relaxing the assumptions of the periodic task model.

The (implicit-deadline) *sporadic* task model relaxed assumption A1 by allowing T_i to be the *minimum* (rather than exact) separation between arrivals of successive jobs of task τ_i. It turns out that the results in Sect. "Results from [9]" – Theorems 1–4 – hold for systems of such tasks as well.

A more general sporadic task model has also been studied that relaxes assumption A3 in addition to assumption A1, by allowing for the explicit specification of a deadline parameter for each task (which may differ from the task's period). The *deadline monotonic* scheduling algorithm [8] generalizes rate-monotonic scheduling to such task systems.

Work has also been done [2,10] in removing the independence assumption of A4, by allowing for different tasks to use critical sections to access non-preemptable serially reusable resources.

Current work is focused on scheduling tasks on multiprocessor or distributed systems where one or more of the assumptions A1–A5 have been relaxed. In addition, recent work has relaxed the assumption (A2) that worst-case execution requirement is known and instead probabilistic execution requirement distributions are considered [4].

Applications

The periodic task model has been invaluable for modeling several different types of systems. For control systems, the periodic task model is well-suited for modeling the periodic requests and computations of sensors and actuators. Multimedia and network applications also typically involve computation of periodically arriving packets and data. Many operating systems for real-time systems provide support for periodic tasks as a standard primitive.

Many of the results described in Sect. "Key Results" above have been integrated into powerful tools, techniques, and methodologies for the design and analysis of real-time application systems [1,5]. Although these are centered around the deadline-monotonic rather than rate-monotonic scheduling algorithm, the general methodology is commonly referred to as the *rate-monotonic analysis* (RMA) methodology.

Open Problems

There are plenty of interesting and challenging open problems in real-time scheduling theory; however, most of these are concerned with extensions to the basic task and scheduling model considered in the original Liu and Layland paper [9]. Perhaps the most interesting open problem with respect to the task model in [9] is regarding the computational complexity of schedulability analysis of static priority scheduling. While all known exact tests (e. g., Theorem 3) run in pseudo-polynomial time and all known polynomial-time tests are sufficient rather than exact, there has been no significant result pigeonholing the computational complexity of static priority schedulability analysis for periodic task systems.

Cross References

▶ List Scheduling
▶ Load Balancing
▶ Schedulers for Optimistic Rate Based Flow Control
▶ Shortest Elapsed Time First Scheduling

Recommended Reading

1. Audsley, N., Burns, A., Wellings, A.: Deadline monotonic scheduling theory and application. Control Eng. Pract. **1**, 71–78 (1993)
2. Baker, T.P.: Stack-based scheduling of real-time processes. Real-Time Systems: The Int. J. Time-Critical Comput. **3**, 67–100 (1991)
3. Bini, E., Buttazzo, G., Buttazzo, G.: Rate monotonic scheduling: The hyperbolic bound. IEEE Trans. Comput. **52**, 933–942 (2003)
4. Gardener, M.K.: Probabilistic Analysis and Scheduling of Critical Soft Real-Time Systems. Ph. D. thesis, Department of Computer Science, University of Illinois at Urbana-Champaign (1999)
5. Klein, M., Ralya, T., Pollak, B., Obenza, R., Harbour, M.G.: A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems. Kluwer Academic Publishers, Boston (1993)
6. Kuo, T.-W., Mok, A.K.: Load adjustment in adaptive real-time systems. In: Proceedings of the IEEE Real-Time Systems Symposium, pp. 160–171. San Antonio, December 1991
7. Lehoczky, J., Sha, L., Ding, Y.: The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In: Proceedings of the Real-Time Systems Symposium – 1989, Santa Monica, December 1989. IEEE Computer Society Press, pp. 166–171
8. Leung, J., Whitehead, J.: On the complexity of fixed-priority scheduling of periodic, real-time tasks. Perform. Eval. **2**, 237–250 (1982)
9. Liu, C., Layland, J.: Scheduling algorithms for multiprogramming in a hard real-time environment. J. ACM **20**, 46–61 (1973)
10. Rajkumar, R.: Synchronization In Real-Time Systems – A Priority Inheritance Approach. Kluwer Academic Publishers, Boston (1991)

Real-Time Systems

▶ Rate-Monotonic Scheduling

Rectilinear Spanning Tree

2002; Zhou, Shenoy, Nicholls

HAI ZHOU
Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL, USA

Keywords and Synonyms

Metric minimum spanning tree; Rectilinear spanning graph

Problem Definition

Given a set of n points in a plane, a spanning tree is a set of edges that connects all the points and contains no cycles. When each edge is weighted using some distance metric of the incident points, the *metric minimum spanning tree* is a tree whose sum of edge weights is minimum. If the Euclidean distance (L_2) is used, it is called the *Euclidean minimum spanning tree*; if the rectilinear distance (L_1) is used, it is called the *rectilinear minimum spanning tree*.

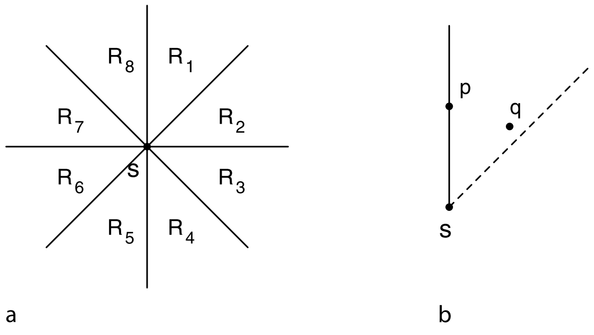
Since the minimum spanning tree problem on a weighted graph is well studied, the usual approach for metric minimum spanning tree is to first define an weighted graph on the set of points and then to construct a spanning tree on it.

Much like a connection graph is defined for the maze search [4], a spanning graph can be defined for the minimum spanning tree construction.

Definition 1 Given a set of points V in a plane, an undirected graph $G = (V, E)$ is called a *spanning graph* if it contains a minimum spanning tree of V in the plane.

Since spanning graphs with fewer edges give more efficient minimum spanning tree construction, the *cardinality* of a spanning graph is defined as its number of edges. It is easy to see that a complete graph on a set of points contains all spanning trees, thus is a spanning graph. However, such a graph has a cardinality of $O(n^2)$. A rectilinear spanning graph of cardinality $O(n)$ can be constructed within $O(n \log n)$ time [6] and will be described here.

Minimum spanning tree algorithms usually use two properties to infer the inclusion and exclusion of edges in a minimum spanning tree. The first property is known as the *cut property*. It states that an edge of smallest weight crossing any partition of the vertex set into two parts belongs to a minimum spanning tree. The second property is known as the *cycle property*. It says that an edge with largest weight in any cycle in the graph can be safely deleted. Since the two properties are stated in connection



Rectilinear Spanning Tree, Figure 1
Octal partition and the uniqueness property

with the construction of a minimum spanning tree, they are useful for a spanning graph.

Key Results

Using the terminology given in [3], the *uniqueness property* is defined as follows.

Definition 2 Given a point s, a region R has the *uniqueness property* with respect to s if for every pair of points $p, q \in R$, $||pq|| < \max(||sp||, ||sq||)$. A partition of space into a finite set of disjoint regions is said to have the uniqueness property with respect to s if each of its regions has the uniqueness property with respect to s.

The notation $||sp||$ is used to represent the distance between s and p under the L_1 metric. Define the *octal partition* of the plane with respect to s as the partition induced by the two rectilinear lines and the two 45 degree lines through s, as shown in Fig. 2a. Here, each of the regions R_1 through R_8 includes only one of its two bounding half line as shown in Fig. 2b. It can be shown that the octal partition has the uniqueness property.

Lemma 1 *Given a point s in the plane, the octal partition with respect to s has the uniqueness property.*

Proof To show a partition has the uniqueness property, it needs to prove that each region of the partition has the uniqueness property. Since the regions R_1 through R_8 are similar to each other, a proof for R_1 will be sufficient.

The points in R_1 can be characterized by the following inequalities

$$x \geq x_s \,,$$
$$x - y < x_s - y_s \,.$$

Suppose there are two points p and q in R_1. Without loss of generality, it can be assumed $x_p \leq x_q$. If $y_p \leq y_q$, then $||sq|| = ||sp|| + ||pq|| > ||pq||$. Therefore it only needs to consider the case when $y_p > y_q$. In this case,

$$\begin{aligned}||pq|| &= |x_p - x_q| + |y_p - y_q| \\ &= x_q - x_p + y_p - y_q \\ &= (x_q - y_q) + y_p - x_p \\ &< (x_s - y_s) + y_p - x_s \\ &= y_p - y_s \\ &\leq x_p - x_s + y_p - y_s \\ &= ||sp|| \,.\end{aligned}$$

□

Given two points p, q in the same octal region of point s, the uniqueness property says that $||pq|| < \max(||sp||, ||sq||)$. Consider the cycle on points s, p, and q. Based on the cycle property, only one point with the minimum distance from s needs to be connected to s. An interesting property of the octal partition is that the contour of equi-distant points from s forms a line segment in each region. In regions R_1, R_2, R_5, R_6, these segments are captured by an equation of the form $x + y = c$; in regions R_3, R_4, R_7, R_8, they are described by the form $x - y = c$.

From each point s, the closest neighbor in each octant needs to be found. It will be described how to efficiently compute the neighbors in R_1 for all points. The case for other octant is symmetric. For the R_1 octant, a sweep line algorithm will run on all points according to non-decreasing $x + y$. During the sweep, maintained will be an *active* set consisting of points whose nearest neighbors in R_1 are yet to be discovered. When a point p is processed, all points in the active set that have p in their R_1 regions will be found. If s is such a point in the active set, since points are scanned in non-decreasing $x + y$, then p must be the nearest point in R_1 for s. Therefore, the edge sp will be added and s will be deleted from the active set. After processing those active points, the point p will be added into the active set. Each point will be added and deleted at most once from the active set.

A fundamental operation in the sweep line algorithm is to find a subset of active points such that a given point p is in their R_1 regions. Based on the observation that point p is in the R_1 region of point s if and only if s is in the R_5 region of p, it needs to find the subset of active points in the R_5 region of p. Since R_5 can be represented as a two-dimensional range $(-\infty, x_p] \times (x_p - y_p, +\infty)$ on $(x, x - y)$, a priority search tree [1] can be used to maintain the active point set. Since each of the insertion and deletion operations takes $O(\log n)$ time, and the query operation takes $O(\log n + k)$ time where k is the number of objects within the range, the total time for the sweep is

$O(n \log n)$. Since other regions can be processed in the similar way as in R_1, the algorithm is running in $O(n \log n)$ time. Priority search tree is a data structure that relies on maintaining a balanced structure for the fast query time. This works well for static input sets. When the input set is dynamic, re-balancing the tree can be quite challenging. Fortunately, the active set has a structure that can be explored for an alternate representation. Since a point is deleted from the active set if a point in its R_1 region is found, no point in the active set can be in the R_1 region of another point in the set.

Lemma 2 *For any two points p, q in the active set, it must be $x_p \neq x_q$, and if $x_p < x_q$ then $x_p - y_p \leq x_q - y_q$.*

Based on this property, the active set can be ordered in increasing order of x. This implies a non-decreasing order on $x - y$. Given a point s, the points which have s in their R_1 region must obey the following inequalities

$$x \leq x_s \,,$$
$$x - y > x_s - y_s \,.$$

To find the subset of active points which have s in their R_1 regions, it can first find the largest x such that $x \leq x_s$, then proceed in decreasing order of x until $x - y \geq x_s - y_s$. Since the ordering is kept on only one dimension, using any binary search tree with $O(\log n)$ insertion, deletion, and query time will also give us an $O(n \log n)$ time algorithm. Binary search trees also need to be balanced. An alternative is to use skip-lists [2] which use randomization to avoid the problem of explicit balancing but provide $O(\log n)$ expected behavior.

A careful study also shows that after the sweep process for R_1, there is no need to do the sweep for R_5, since all edges needed in that phase are either connected or implied. Moreover, based on the information in R_5, the number of edge connections can be further reduced. When the sweep step processes point s, it finds a subset of active points which have s in their R_1 regions. Without lost of generality, suppose p and q are two of them. Then p and q are in the R_5 region of s, which means $||pq|| < \max(||sp||, ||sq||)$. Therefore, it needs only to connect s with the nearest active point.

Since R_1 and R_2 have the same sweep sequence, they can be processed together in one pass. Similarly, R_3 and R_4 can be processed together in another pass. Based on the above discussion, the pseudo-code of the algorithm is presented in Fig. 2.

The correctness of the algorithm is stated in the following theorem.

```
Rectilinear Spanning Graph Algorithm
  for $(i = 0; i < 2; i++)$ {
    if $(i == 0)$ sort points according to $x + y$;
    else sort points according to $x - y$;
    $A[1] = A[2] = \emptyset$;
    for each point $p$ in the order {
      find points in $A[1], A[2]$ such that $p$ is in their
        $R_{2i+1}$ and $R_{2i+2}$ regions, respectively;
      connect $p$ with the nearest point in each subset;
      delete the subsets from $A[1], A[2]$, respectively;
      add $p$ to $A[1], A[2]$;
    }
  }
```

Rectilinear Spanning Tree, Figure 2
The rectilinear spanning graph algorithm

Theorem 3 *Given n points in the plane, the rectilinear spanning graph algorithm constructs a spanning graph in $O(n \log n)$ time, and the number of edges in the graph is $O(n)$.*

Proof The algorithm can be considered as deleting edges from the complete graph. As described, all deleted edges are redundant based on the cycle property. Thus, the output graph of the algorithm will contain at least one rectilinear minimum spanning tree.

In the algorithm, each given point will be inserted and deleted at most once from the active set for each of the four regions R_1 through R_4. For each insertion or deletion, the algorithm requires $O(\log n)$ time. Thus, the total time is upper bounded by $O(n \log n)$. The storage is needed only for active sets, which is at most $O(n)$. □

Applications

Rectilinear minimum spanning tree problem has wide applications in VLSI CAD. It is frequently used as a metric of wire length estimation during placement. It is often constructed to approximate a minimum Steiner tree and is also a key step in many Steiner tree heuristics. It is also used in an approximation to the traveling salesperson problem which can be used to generate scan chains in testing. It is important to emphasize that for real world applications, the input sizes are usually very large. Since it is a problem that will be computed hundreds of thousands times and many of them will have very large input sizes, the rectilinear minimum spanning tree problem needs a very efficient algorithm.

Experimental Results

The experimental results using the Rectilinear Spanning Graph (RSG) followed by Kruskal's algorithm for a rectilinear minimum spanning tree were reported in Zhou

Rectilinear Spanning Tree, Table 1
Experimental Results

Input		Complete		Bound-degree		RSG	
orig	distinct	#edge	time	#edge	time	#edge	time
1000	999	498501	5.095 s	3878	0.299 s	2571	0.112 s
2000	1996	1991010	24.096 s	7825	0.996 s	5158	0.218 s
4000	3995	7978015	2 m 7.233 s	15761	3.452 s	10416	0.337 s
6000	5991	17943045	5 m 54.697 s	23704	7.515 s	15730	0.503 s
8000	7981	31844190	13 m 7.682 s	31624	13.141 s	21149	0.672 s
10000	9962	49615741	–	39510	20.135 s	26332	0.934 s
12000	11948	–	–	47424	32.300 s	31586	1.052 s
14000	13914	–	–	55251	46.842 s	36853	1.322 s
16000	15883	–	–	63089	1 m 3.759 s	42251	1.486 s
18000	17837	–	–	70876	1 m 19.812 s	47511	1.701 s
20000	19805	–	–	78723	1 m 45.792 s	52732	1.907 s

et al. [5]. Two other approaches were compared. The first approach used the complete graph on the point set as the input to Kruskal's algorithm. The second approach is an implementation of concepts described in [3]; namely for each point, scan all other points but only connect the nearest one in each quadrant region. With sizes ranging from 1000 to 20,000, randomly generated point sets were used in the experiments. The results are reproduced here in Table 1. The first column gives the number of generated points; the second column gives the number of distinct points. For each approach, the number of edges in the given graph and the total running time are reported. For input size larger than 10,000, the complete graph approach simply runs out of memory.

Cross References

► Rectilinear Steiner Tree

Recommended Reading

1. McCreight, E.M.: Priority search trees. SIAM J. Comput. **14**, 257–276 (1985)
2. Pugh, W.: Skip lists: A probabilistic alternative to balanced trees. Commun. ACM **33**, 668–676 (1990)
3. Robins, G., Salowe, J.S.: Low-degree minimum spanning tree. Discret. Comput. Geom. **14**, 151–165 (1995)
4. Zheng, S.Q., Lim, J.S., Iyengar, S.S.: Finding obstacle-avoiding shortest paths using implicit connection graphs. IEEE Trans. Comput. Aided Des. **15**, 103–110 (1996)
5. Zhou, H., Shenoy, N., Nicholls, W.: Efficient minimum spanning tree construction without delaunay triangulation. In: Proc. Asian and South Pacific Design Automation Conference, Yokohama, Japan (2001)
6. Zhou, H., Shenoy, N., Nicholls, W.: Efficient spanning tree construction without delaunay triangulation. Inf. Proc. Lett. **81**, 271–276 (2002)

Rectilinear Steiner Tree

2004; Zhou

HAI ZHOU
Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL, USA

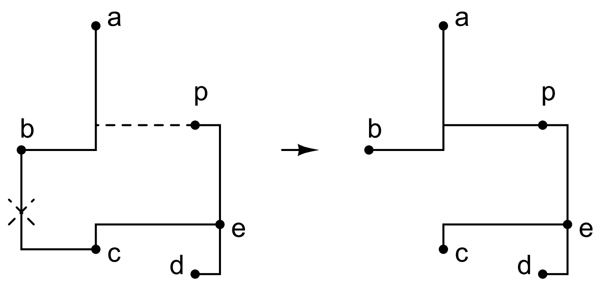
Keywords and Synonyms

Metric minimum Steiner tree; Shortest routing tree

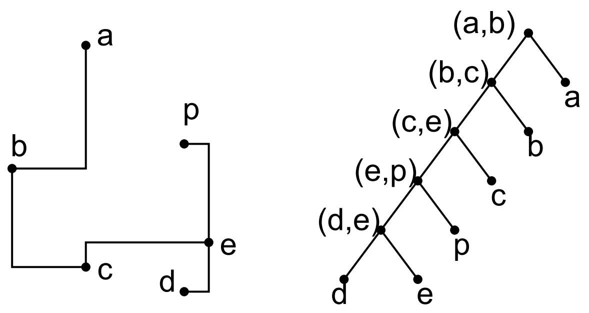
Problem Definition

Given *n* points on a plane, a Steiner minimal tree connects these points through some extra points (called Steiner points) to achieve a minimal total length. When the length between two points is measured by the rectilinear distance, the tree is called a rectilinear Steiner minimal tree.

Because of its importance, there is much previous work to solve the SMT problem. These algorithms can be grouped into two classes: exact algorithms and heuristic algorithms. Since SMT is NP-hard, any exact algorithm is expected to have an exponential worst-case running time. However, two prominent achievements must be noted in this direction. One is the `GeoSteiner` algorithm and implementation by Warme, Winter, and Zacharisen [14,15], which is the current fastest exact solution to the problem. The other is a Polynomial Time Approximation Scheme (PTAS) by Arora [1], which is mainly of theoretical importance. Since exact algorithms have long running time, especially on large input sizes, much more previous efforts were put on heuristic algorithms. Many of them generate a Steiner tree by improving on a minimal spanning tree topology [7], since it was



Rectilinear Steiner Tree, Figure 1
Edge substitution by Borah et al.



Rectilinear Steiner Tree, Figure 2
A minimal spanning tree and its merging binary tree

proved that a minimal spanning tree is a 3/2 approximation of a SMT [8]. However, since the backbones are restricted to the minimal spanning tree topology in these approaches, there is a reported limit on the improvement ratios over the minimal spanning trees. The iterated 1-Steiner algorithm by Kahng and Robins [10] is an early approach to deviate from that restriction and an improved implementation [6] is a champion among such programs in public domain. However, the implementation in [10] has a running time of $O(n^4 \log n)$ and the implementation in [6] has a running time of $O(n^3)$. A much more efficient approach was later proposed by Borah et al. [2]. In their approach, a spanning tree is iteratively improved by connecting a point to an edge and deleting the longest edge on the created circuit. Their algorithm and implementation had a worst-case running time of $\Theta(n^2)$, even though an alternative $O(n \log n)$ implementation was also proposed. Since the backbone is no longer restricted to the minimal spanning tree topology, its performance was reported to be similar to the iterated 1-Steiner algorithm [2]. A recent effort in this direction is a new heuristic by Mandoiu et al. [11] which is based on a 3/2 approximation algorithm of the metric Steiner tree problem on quasi-bipartite graphs [12]. It performs slightly better than the iterated 1-Steiner algorithm, but its running time is also slightly longer than the iterated 1-Steiner algorithm (with the empty rectangle test [11] used). More recently, Chu [3] and Chu and Wong [4] proposed an efficient lookup table based approach for rectilinear Steiner tree construction.

Key Results

The presented algorithm is based on the edge substitution heuristic of Borah et al. [2]. The heuristic works as follows. It starts with a minimal spanning tree and then iteratively considers connecting a point (for example p in Fig. 1) to a nearby edge (for example (a, b)) and deleting the longest edge $((b, c))$ on the circuit thus formed. The algorithm employs the spanning graph [17] as a backbone of the computation: it is first used to generate the initial minimal spanning tree, and then to generate point-edge pairs for tree improvements. This kind of unification happens also in the spanning tree computation and the longest edge computation for each point-edge pair: using Kruskal's algorithm with disjoint set operations (instead of Prim's algorithm) [5] will unify these two computations.

In order to reduce the number of point-edge pair candidates from $O(n^2)$ to $O(n)$, Borah et al. suggested to use the visibility of a point from an edge, that is, only a point visible from an edge can be considered to connect to that edge. This requires a sweepline algorithm to find visibility relations between points and edges. In order to skip this complex step, the geometrical proximity information embedded within the spanning graph is leveraged. Since a point has eight nearest points connected around it, it is observed that if a point is visible to an edge then the point is *usually* connected in the graph to at lease one end point. In the algorithm, the spanning graph is used to generate point-edge pair candidates. For each edge in the current tree, all points that are neighbors of either of the end points will be considered to form point-edge pairs with the edge. Since the cardinality of the spanning graph is $O(n)$, the number of possible point-edge pairs generated in this way is also $O(n)$.

When connecting a point to an edge, the longest edge on the formed circuit needs to be deleted. In order to find the corresponding longest edge for each point-edge pair efficiently, it explores how the spanning tree is formed through Kruskal's algorithm. This algorithm first sorts the edges into non-decreasing lengths and each edge is considered in turn. If the end points of the edge have been connected, then the edge will be excluded from the spanning tree, otherwise, it will be included. The structure of these connecting operations can be represented by a binary tree, where the leaves represent the points and the

```
Rectilinear Steiner Tree (RST) Algorithm
    $T = \emptyset$;
    Generate the spanning graph $G$ by RSG algorithm;
    for (each edge $(u, v) \in G$ in non-decreasing length) {
        $s1$ = find_set($u$); $s2$ = find_set($v$);
        if ($s1$ != $s2$) {
            add $(u, v)$ in tree $T$;
            for (each neighbor $w$ of $u, v$ in $G$)
                if ($s1$ == find_set($w$))
                    lca_add_query($w, u, (u, v)$);
                else lca_add_query($w, v, (u, v)$);
            lca_tree_edge($(u, v)$, $s1$.edge);
            lca_tree_edge($(u, v)$, $s2$.edge);
            $s$ = union_set($s1, s2$); $s$.edge = $(u, v)$;
        }
    }
    generate point-edge pairs by lca_answer_queries;
    for (each pair $(p, (a, b), (c, d))$ in non-increasing positive gains)
        if ($(a, b), (c, d)$ has not been deleted from $T$) {
            connect $p$ to $(a, b)$ by adding three edges to $T$;
            delete $(a, b), (c, d)$ from $T$;
        }
```

Rectilinear Steiner Tree, Figure 3
The rectilinear Steiner tree algorithm

internal nodes represent the edges. When an edge is included in the spanning tree, a node is created representing the edge and has as its two children the trees representing the two components connected by this edge. To illustrate this, a spanning tree with its representing binary tree are shown in Fig. 2. As can be seen, the longest edge between two points is the least common ancestor of the two points in the binary tree. For example, the longest edge between p and b in Fig. 2 is (b, c), which is the least common ancestor of p and b in the binary tree. To find the longest edge on the circuit formed by connecting a point to an edge, it needs to find the longest edge between the point and one end point of the edge that are in the same component before connecting the edge. For example, consider the pair p and (a, b), since p and b are in the same component before connecting (a, b), the edge needs to be deleted is the longest between p and b.

Based on the above discussion, the pseudo-code of the algorithm can be described in Fig. 3. At the beginning of the algorithm, Zhou et al.'s rectilinear spanning graph algorithm [17] is used to generate the spanning graph G for the given set of points. Then Kruskal's algorithm is used on the graph to generate a minimal spanning tree. The data structure of disjoint sets [5] is used to merge components and check whether two points are in the same component (the first **for** loop). During this process, the merging binary tree and the queries for least common ancestors of all point-edge pairs are also generated. Here s, $s1$, and $s2$ represent disjoint sets and each records the root of the component in the merging binary tree. For each edge (u, v) adding to T, each neighbor w of either u or v will be considered to connect to (u, v). The longest edge for this pair is the least common ancestor of w, u or w, v depending on which point is in the same component as w. The procedure `lca_add_query` is used to add this query. Connecting the two components by (u, v) will also be recorded in the merging binary tree by the procedure `lca_tree_edge`. After generating the minimal spanning tree, it also has the corresponding merging binary tree and the least common ancestor queries ready. Using Tarjan's off-line least common ancestor algorithm [5] (represented by `lca_answer_queries`), it can generate all longest

Rectilinear Steiner Tree, Table 1
Comparison with other algorithms I

Input size	GeoSteiner		BI1S		BOI		RST	
	Improve	Time	Improve	Time	Improve	Time	Improve	Time
100	11.440	0.487	10.907	0.633	9.300	0.0267	10.218	0.004
200	11.492	3.557	10.897	4.810	9.192	0.1287	10.869	0.020
300	11.492	12.685	10.931	18.770	9.253	0.2993	10.255	0.041
500	11.525	72.192	–	–	9.274	0.877	10.381	0.084
800	11.343	536.173	–	–	9.284	2.399	10.719	0.156
1000	–	–	–	–	9.367	4.084	10.433	0.186
2000	–	–	–	–	9.326	31.098	10.523	0.381
3000	–	–	–	–	9.390	104.919	10.449	0.771
5000	–	–	–	–	9.356	307.977	10.499	1.330

Rectilinear Steiner Tree, Table 2
Comparison with other algorithms II

Input size	BGA		Borah		Rohe		RST	
	Improve	Time	Improve	Time	Improve	Time	Improve	Time
Randomly generated testcases								
100	10.272	0.006	10.341	0.004	9.617	0.000	10.218	0.002
500	10.976	0.068	10.778	0.178	10.028	0.010	10.381	0.041
1000	10.979	0.162	10.829	0.689	9.768	0.020	10.433	0.121
5000	11.012	1.695	11.015	25.518	10.139	0.130	10.499	0.980
10000	11.108	4.135	11.101	249.924	10.111	0.310	10.559	2.098
50000	11.120	59.147	–	–	10.109	1.890	10.561	13.029
100000	11.098	161.896	–	–	10.079	4.410	10.514	28.527
500000	–	–	–	–	10.059	27.210	10.527	175.725
VLSI testcases								
337	6.434	0.035	6.503	0.037	5.958	0.010	5.870	0.016
830	3.202	0.070	3.185	0.213	3.102	0.020	2.966	0.033
1944	7.850	0.342	7.772	2.424	6.857	0.040	7.533	0.238
2437	7.965	0.549	7.956	4.502	7.094	0.050	7.595	0.408
2676	8.928	0.623	8.994	3.686	8.067	0.060	8.507	0.463
12052	8.450	4.289	8.465	232.779	7.649	0.300	8.076	2.281
22373	9.848	11.330	9.832	1128.365	8.987	0.570	9.462	4.605
34728	9.046	18.416	9.010	2367.629	8.158	0.900	8.645	5.334

edges for the pairs. With the longest edge for each point-edge pair, the gain of connecting the point to the edge can be calculated. Then each of the point to edge connections will be realized in a non-increasing order of their gains. A connection can only be realized if both the connection edge and deletion edge have not been deleted yet.

The running time of the algorithm is dominated by the spanning graph generation and edge sorting, which take $O(n \log n)$ time. Since the number of edges in the spanning graph is $O(n)$, both Kruskal's algorithm and Tarjan's off-line least common ancestor algorithm take $O(n\alpha(n))$ time, where $\alpha(n)$ is the inverse of Ackermann's function, which grows extremely slow.

Applications

The Steiner Minimal Tree (SMT) problem has wide applications in VLSI CAD. A SMT is generally used in initial topology creation for non-critical nets in physical synthesis. For timing critical nets, minimization of wire length is generally not enough. However, since most nets are non-critical in a design and a SMT gives the most desirable route of such a net, it is often used as an accurate estimation of congestion and wire length during floorplanning and placement. This implies that a Steiner tree algorithm will be invoked millions of times. On the other hand, there exist many large pre-routes in modern VLSI design. The pre-routes are generally modeled as large sets of points, thus increasing the input sizes of the Steiner tree problem. Since the SMT is a problem that will be computed millions of times and many of them will have very large input sizes, highly efficient solutions with good performance are desired.

Experimental Results

As reported in [16], the first set of experiments were conducted on a Linux system with a 928 MHz Intel Pentium III processor and 512 M memory. The RST algorithm was compared with other publicly available programs: the exact algorithm GeoSteiner (version 3.1) by Warme, Winter, and Zacharisen [14]; the Batched Iterated 1-Steiner (BI1S) by Robins; and the Borah et al.'s algorithm implemented by Madden (BOI).

Table 1 gives the results of the first set of experiments. For each input size ranging from 100 to 5000, 30 different test cases are randomly generated through the rand_points program in GeoSteiner. The improvement ratios of a Steiner tree St over its corresponding minimal spanning tree MST is defined as $100 \times (\mathtt{MST} - \mathtt{St})/\mathtt{MST}$. For each input size, the average of the improvement ratios and the average running time (in seconds) on each of the programs is reported. As can be seen, RST always gives better improvements than BOI with less running times.

The second set of experiments compared RST with Borah's implementation of Borah et al.'s algorithm (Borah), Rohe's Prim-based algorithm (Rohe) [13], and Kahng et al.'s Batched Greedy Algorithm (BGA) [9]. They were run on a different Linux system with a 2.4 GHz Intel Xeon processor and 2 G memory. Besides the randomly generated test cases, the VLSI industry test cases used in [9] were also used. The results are reported in Table 2.

Cross References

► Rectilinear Spanning Tree

Recommended Reading

1. Arora, S.: Polynomial-time approximation schemes for euclidean tsp and other geometric problem. J. ACM **45**, 753–782 (1998)
2. Borah, M., Owens, R.M., Irwin, M.J.: An edge-based heuristic for steiner routing. IEEE Transac. Comput. Aided Des. **13**, 1563–1568 (1994)
3. Chu, C.: FLUTE: Fast lookup table based wirelength estimation technique. In: Proc. Intl. Conf. on Computer-Aided Design, San Jose, Nov. 2004, pp. 696–701
4. Chu, C., Wong, Y.C.: Fast and accurate rectilinear steiner minimal tree algorithm for vlsi design. In: International Symposium on Physical Design, pp. 28–35 (2005)
5. Cormen, T.H., Leiserson, C.E., Rivest, R.L.: Introduction to Algorithms. MIT Press, Cambridge (1989)
6. Griffith, J., Robins, G., Salowe, J.S., Zhang, T.: Closing the gap: Near-optimal steiner trees in polynomial time. IEEE Transac. Comput. Aided Des. **13**, 1351–1365 (1994)
7. Ho, J.M., Vijayan, G., Wong, C.K.: New algorithms for the rectilinear steiner tree problem. IEEE Transac. Comput. Aided Des. **9**, 185–193 (1990)
8. Hwang, F.K.: On Steiner minimal trees with rectilinear distance. SIAM J. Appl. Math. **30**, 104–114 (1976)
9. Kahng, A.B., Mandoiu, I.I., Zelikovsky, A.: Highly scalable algorithms for rectilinear and octilinear steiner trees. In: Proc. Asia and South Pacific Design Automation Conference, Kitakyushu, Japan, (2003) pp. 827–833
10. Kahng, A.B., Robins, G.: A new class of iterative steiner tree heuristics with good performance. IEEE Transac. Comput. Aided Des. **11**, 893–902 (1992)
11. Mandoiu, I.I., Vazirani, V.V., Ganley, J.L.: A new heuristic for rectilinear Steiner trees. In: Proc. Intl. Conf. on Computer-Aided Design, San Jose, (1999)
12. Rajagopalan, S., Vazirani, V.V.: On the bidirected cut relaxation for the metric Steiner tree problem. In: 10th ACM-SIAM Symposium on Discrete Algorithms, Baltimore, (1999), pp. 742–751
13. Rohe, A.: Sequential and Parallel Algorithms for Local Routing. Ph. D. thesis, Bonn University, Bonn, Germany, Dec. (2001)
14. Warme, D.M., Winter, P., Zacharisen, M.: GeoSteiner 3.1 package. ftp://ftp.diku.dk/diku/users/martinz/geosteiner-3.1.tar.gz. Accessed Oct. 2003
15. Warme, D.M., Winter, P., Zacharisen, M.: Exact algorithms for plane steiner tree problems: A computational study, Tech. Rep. DIKU-TR-98/11, Dept. of Computer Science, University of Copenhagen (1998)
16. Zhou, H.: A new efficient retiming algorithm derived by formal manipulation. In: Workshop Notes of Intl. Workshop on Logic Synthesis, Temecula, CA, June (2004)
17. Zhou, H., Shenoy, N., Nicholls, W.: Efficient spanning tree construction without delaunay triangulation. Inf. Process. Lett. **81**, 271–276 (2002)

Registers

1986; Lamport, Vitanyi, Awerbuch

PAUL VITÁNYI
CWI, Amsterdam, Netherlands

Keywords and Synonyms

Shared-memory (wait-free); Wait-free registers; Wait-free shared variables; Asynchronous communication hardware

Problem Definition

Consider a system of asynchronous processes that communicate among themselves by only executing read and write operations on a set of shared variables (also known as shared *registers*). The system has no global clock or other synchronization primitives. Every shared variable is associated with a process (called *owner*) which writes it and the other processes may read it. An execution of a write (read) operation on a shared variable will be referred to as a *Write* (*Read*) on that variable. A Write on a shared variable puts a value from a pre-determined finite domain into the variable, and a Read reports a value from the domain. A process that writes (reads) a variable is called a *writer* (*reader*) of the variable.

The goal is to construct shared variables in which the following two properties hold. (1) Operation executions are not necessarily atomic, that is, they are not indivisible but rather consist of atomic sub-operations, and (2) every operation finishes its execution within a bounded number of its own steps, irrespective of the presence of other operation executions and their relative speeds. That is, operation executions are *wait-free*. These two properties give rise to a classification of shared variables, depending on their output characteristics. Lamport [8] distinguishes three categories for 1-writer shared variables, using a precedence relation on operation executions defined as follows: for operation executions A and B, A *precedes* B, denoted $A \longrightarrow B$, if A finishes before B starts; A and B *overlap* if neither A precedes B nor B precedes A. In 1-writer variables, all the Writes are totally ordered by "$\longrightarrow$". The three categories of 1-writer shared variables defined by Lamport are the following.

1. A *safe* variable is one in which a Read not overlapping any Write returns the most recently written value. A Read that overlaps a Write may return any value from the domain of the variable.
2. A *regular* variable is a safe variable in which a Read that overlaps one or more Writes returns either the value of the most recent Write preceding the Read or of one of the overlapping Writes.
3. An *atomic* variable is a regular variable in which the Reads and Writes behave as if they occur in some total order which is an extension of the precedence relation.

A shared variable is *boolean*[1] or *multivalued* depending upon whether it can hold only one out of two or one out of more than two values. A *multiwriter* shared variable is one that can be written and read (concurrently) by many processes. If there is only one writer and more than one reader it is called a *multireader* variable.

Key Results

In a series of papers starting in 1974, for details see [4], Lamport explored various notions of concurrent reading and writing of shared variables culminating in the seminal 1986 paper [8]. It formulates the notion of wait-free implementation of an atomic multivalued shared variable—written by a single writer and read by (another) single reader—from safe 1-writer 1-reader 2-valued shared variables, being mathematical versions of physical *flip-flops*, later optimized in [13]. Lamport did not consider constructions of shared variables with more than one writer or reader.

Predating the Lamport paper, in 1983 Peterson [10] published an ingenious wait-free construction of an atomic 1-writer, n-reader m-valued atomic shared variable from $n + 2$ safe 1-writer n-reader m-valued registers, $2n$ 1-writer 1-reader 2-valued atomic shared variables, and 2 1-writer n-reader 2-valued atomic shared variables. He presented also a proper notion of the wait-freedom property. In his paper, Peterson didn't tell how to construct the n-reader boolean atomic variables from flip-flops, while Lamport mentioned the open problem of doing so, and, incidentally, uses a version of Peterson's construction to bridge the algorithmically demanding step from atomic shared bits to atomic shared multivalues. On the basis of this work, N. Lynch, motivated by concurrency control of multi-user data-bases, posed around 1985 the question of how to construct wait-free multiwriter atomic variables from 1-writer multireader atomic variables. Her student Bloom [1] found in 1985 an elegant 2-writer construction, which, however, has resisted generalization to multiwriter. Vitányi and Awerbuch [14] were the first to define and explore the complicated notion of wait-free constructions of general multiwriter atomic variables, in 1986. They presented a proof method, an unbounded solution from 1-writer 1-reader atomic variables, and a bounded solution from 1-writer n-reader atomic variables. The bounded solution turned out not to be atomic, but only achieved regularity ("Errata" in [14]). The paper introduced important notions and techniques in the area, like (bounded) vector clocks, and identified open problems like the construction of atomic wait-free bounded multireader shared variables from flip-flops, and atomic wait-free bounded multiwriter shared variables from the multireader ones. Peterson who had been working on the multiwriter problem for a decade, together with Burns, tried in 1987 to eliminate the error in the unbounded construction of [14] retaining the idea of vector clocks, but replacing the obsolete-information tracking technique by repeated scanning as in [10]. The result [11] was found to be erroneous in the technical report (R. Schaffer, On the correctness of atomic multiwriter registers, Report MIT/LCS/TM-364, 1988). Neither the re-correction in Schaffer's Technical Report, nor the claimed re-correction by the authors of [11] has appeared in print.

[1] Boolean variables are referred to as *bits*.

Also in 1987 there appeared at least five purported solutions for the implementation of 1-writer n-reader atomic shared variable from 1-writer 1-reader ones: [2,7,12] (for the others see [4]) of which [2] was shown to be incorrect (S. Haldar, K. Vidyasankar, *ACM Oper. Syst. Rev*, 26:1(1992), 87–88) and only [12] appeared in journal version. The paper [9], initially a 1987 Harvard Tech Report, resolved all multiuser constructions in one stroke: it constructs a bounded n-writer n-reader (multiwriter) atomic variable from $O(n^2)$ 1-writer 1-reader safe bits, which is optimal, and $O(n^2)$ bit-accesses per Read/Write operation which is optimal as well. It works by making the unbounded solution of [14] bounded, using a new technique, achieving a robust proof of correctness. "Projections" of the construction give specialized constructions for the implementation of 1-writer n-reader (multireader) atomic variables from $O(n^2)$ 1-writer 1-reader ones using $O(n)$ bit accesses per Read/Write operation, and for the implementation of n-writer n-reader (multiwriter) atomic variables from n 1-writer n-reader (multireader) ones. The first "projection" is optimal, while the last "projection" may not be optimal since it uses $O(n)$ control bits per writer while only a lower bound of $\Omega(\log n)$ was established. Taking up this challenge, the construction in [6] claims to achieve this lower bound.

Timestamp System

In a multiwriter shared variable it is only required that every process keeps track of which process wrote last. There arises the general question whether every process can keep track of the order of the last Writes by all processes. A. Israeli and M. Li were attracted to the area by the work in [14], and, in an important paper [5], they raised and solved the question of the more general and universally useful notion of a bounded timestamp system to track the order of events in a concurrent system. In a timestamp system every process owns an *object*, an abstraction of a set of shared variables. One of the requirements of the system is to determine the temporal order in which the objects are written. For this purpose, each object is given a *label* (also referred to as a *timestamp*) which indicates the latest (relative) time when it has been written by its owner process. The processes assign labels to their respective objects in such a way that the labels reflect the real-time order in which they are written to. These systems must support two operations, namely *labeling* and *scan*. A labeling operation execution (Labeling, in short) assigns a new label to an object, and a scan operation execution (Scan, in short) enables a process to determine the ordering in which all the objects are written, that is, it returns a set of labeled-objects ordered temporally. The concern is with those systems where operations can be executed *concurrently*, in an overlapped fashion. Moreover, operation executions must be *wait-free*, that is, each operation execution will take a bounded number of its own steps (the number of accesses to the shared space), irrespective of the presence of other operation executions and their relative speeds. Israeli and Li [5] constructed a bit-optimal bounded timestamp system for *sequential* operation executions. Their sequential timestamp system was published in the above journal reference, but the preliminary concurrent timestamp system in the conference proceedings, of which a more detailed version has been circulated in manuscript form, has not been published in final form. The first generally accepted solution of the *concurrent* case of the bounded timestamp system was from Dolev and Shavit [3]. Their construction is of the type presented in [5] and uses shared variables of size $O(n)$, where n is the number of processes in the system. Each Labeling requires $O(n)$ steps, and each Scan $O(n^2 \log n)$ steps. (A 'step' accesses an $O(n)$ bit variable.) In [4] the unbounded construction of [14] is corrected and extended to obtain an efficient version of the more general notion of a bounded concurrent timestamp system.

Applications

Wait-free registers are, together with message-passing systems, the primary interprocess communication method in distributed computing theory. They form the basis of all constructions and protocols, as can be seen in the textbooks. Wait-free constructions of concurrent timestamp systems (CTSs, in short) have been shown to be a powerful tool for solving concurrency control problems such as various types of mutual exclusion, multiwriter multireader shared variables [14], and probabilistic consensus, by synthesizing a "wait-free clock" to sequence the actions in a concurrent system. For more details see [4].

Open Problems

There is a great deal of work in the direction of register constructions that use less constituent parts, or simpler parts, or parts that can tolerate more complex failures, than previous constructions referred to above. Only, of course, if the latter constructions were not yet optimal in the parameter concerned. Further directions are work on wait-free higher-typed objects, as mentioned above, hierarchies of such objects, and probabilistic constructions. This literature is too vast and diverse to be surveyed here.

Experimental Results

Register constructions, or related constructions for asynchronous interprocess communication, are used in current hardware and software.

Cross References

- ▶ Asynchronous Consensus Impossibility
- ▶ Atomic Broadcast
- ▶ Causal Order, Logical Clocks, State Machine Replication
- ▶ Concurrent Programming, Mutual Exclusion
- ▶ Linearizability
- ▶ Renaming
- ▶ Self-Stabilization
- ▶ Snapshots in Shared Memory
- ▶ Synchronizers, Spanners
- ▶ Topology Approach in Distributed Computing

Recommended Reading

1. Bloom, B.: Constructing two-writer atomic registers. IEEE Trans. Comput. **37**(12), 1506–1514 (1988)
2. Burns, J.E., Peterson, G.L.: Constructing multi-reader atomic values from non-atomic values. In: Proc. 6th ACM Symp. Principles Distr. Comput., pp. 222–231. Vancouver, 10–12 August 1987
3. Dolev, D., Shavit, N.: Bounded concurrent time-stamp systems are constructible. SIAM J. Comput. **26**(2), 418–455 (1997)
4. Haldar, S., Vitanyi, P.: Bounded concurrent timestamp systems using vector clocks. J. Assoc. Comp. Mach. **49**(1), 101–126 (2002)
5. Israeli, A., Li, M.: Bounded time-stamps. Distribut. Comput. **6**, 205–209 (1993) (Preliminary, more extended, version in: Proc. 28th IEEE Symp. Found. Comput. Sci., pp. 371–382, 1987.)
6. Israeli, A., Shaham, A.: Optimal multi-writer multireader atomic register. In: Proc. 11th ACM Symp. Principles Distr. Comput., pp. 71–82. Vancouver, British Columbia, Canada, 10–12 August 1992
7. Kirousis, L.M., Kranakis, E., Vitányi, P.M.B.: Atomic multireader register. In: Proc. Workshop Distributed Algorithms. Lect Notes Comput Sci, vol 312, pp. 278–296. Springer, Berlin (1987)
8. Lamport, L.: On interprocess communication—Part I: Basic formalism, Part II: Algorithms. Distrib. Comput. **1**(2), 77–101 (1986)
9. Li, M., Tromp, J., Vitányi, P.M.B.: How to share concurrent wait-free variables. J. ACM **43**(4), 723–746 (1996) (Preliminary version: Li, M., Vitányi, P.M.B. A very simple construction for atomic multiwriter register. Tech. Rept. TR-01–87, Computer Science Dept., Harvard University, Nov. 1987)
10. Peterson, G.L.: Concurrent reading while writing. ACM Trans. Program. Lang. Syst. **5**(1), 56–65 (1983)
11. Peterson, G.L., Burns, J.E.: Concurrent reading while writing II: The multiwriter case. In: Proc. 28th IEEE Symp. Found. Comput. Sci., pp. 383–392. Los Angeles, 27–29 October 1987
12. Singh, A.K., Anderson, J.H., Gouda, M.G.: The elusive atomic register. J. ACM **41**(2), 311–339 (1994) (Preliminary version in: Proc. 6th ACM Symp. Principles Distribt. Comput., 1987)
13. Tromp, J.: How to construct an atomic variable. In: Proc. Workshop Distrib. Algorithms. Lecture Notes in Computer Science, vol. 392, pp. 292–302. Springer, Berlin (1989)
14. Vitányi, P.M.B., Awerbuch, B.: Atomic shared register access by asynchronous hardware. In: Proc. 27th IEEE Symp. Found. Comput. Sci. pp. 233–243. Los Angeles, 27–29 October 1987. Errata, Proc. 28th IEEE Symp. Found. Comput. Sci., pp. 487–487. Los Angeles, 27–29 October 1987

Regular Expression Indexing

2002; Chan, Garofalakis, Rastogi

CHEE-YONG CHAN[1], MINOS GAROFALAKIS[2], RAJEEV RASTOGI[3]

[1] Department of Computer Science, National University of Singapore, Singapore, Singapore

[2] Computer Science Division, University of California – Berkeley, Berkeley, CA, USA

[3] Bell Labs, Lucent Technologies, Murray Hill, NJ, USA

Keywords and Synonyms

Regular expression indexing; Regular expression retrieval

Problem Definition

Regular expressions (REs) provide an expressive and powerful formalism for capturing the structure of messages, events, and documents. Consequently, they have been used extensively in the specification of a number of languages for important application domains, including the XPath pattern language for XML documents [6], and the policy language of the *Border Gateway Protocol* (BGP) for propagating routing information between autonomous systems in the Internet [12]. Many of these applications have to manage large databases of RE specifications and need to provide an effective matching mechanism that, given an input string, quickly identifies all the REs in the database that match it. This RE retrieval problem is therefore important for a variety of software components in the middleware and networking infrastructure of the Internet.

The RE retrieval problem can be stated as follows: Given a large set S of REs over an alphabet Σ, where each RE $r \in S$ defines a regular language $L(r)$, construct a data structure on S that efficiently answers the following query: given an arbitrary input string $w \in \Sigma^*$, find the subset S_w of REs in S whose defined regular languages include the string w. More precisely, $r \in S_w$ iff $w \in L(r)$. Since S is a large, dynamic, disk-resident collection of REs, the data structure should be dynamic and provide efficient support of updates (insertions and deletions) to S. Note that this problem is the opposite of the more traditional RE search problem where $S \subseteq \Sigma^*$ is a collection of strings and the task is to efficiently find all strings in S that match an input regular expression.

Notations

An RE r over an alphabet Σ represents a subset of strings in Σ^* (denoted by $L(r)$) that can be defined recursively as follows [9]: (1) the constants ϵ and $\emptyset$ are REs, where $L(\epsilon) = \{\epsilon\}$ and $L(\emptyset) = \emptyset$; (2) for any letter $a \in \Sigma$, a is a RE where $L(a) = \{a\}$; (3) if r_1 and r_2 are REs, then their union, denoted by $r_1 + r_2$, is a RE where $L(r_1 + r_2) = L(r_1) \cup L(r_2)$; (4) if r_1 and r_2 are REs, then their concatenation, denoted by $r_1.r_2$, is a RE where $L(r_1.r_2) = \{s_1 s_2 \mid s_1 \in L(r_1), s_2 \in L(r_2)\}$; (5) if r is a RE, then its closure, denoted by r^*, is a RE where $L(r^*) = L(\epsilon) \cup L(r) \cup L(rr) \cup L(rrr) \cup \cdots$; and (6) if r is a RE, then a parenthesized r, denoted by (r), is a RE where $L((r)) = L(r)$. For example, if $\Sigma = \{a, b, c\}$, then $(a + b).(a + b + c)^*.c$ is a RE representing the set of strings that begins with either a "a" or a "b" and ends with a "c". A string $s \in \Sigma^*$ is said to match a RE r if $s \in L(r)$.

The language $L(r)$ defined by an RE r can be recognized by a *finite automaton (FA) M* that decides if an input string w is in $L(r)$ by reading each letter in w sequentially and updating its current state such that the outcome is determined by the final state reached by M after w has been processed [9]. Thus, M is an FA for r if the language accepted by M, denoted by $L(M)$, is equal to $L(r)$. An FA is classified as a *deterministic finite automaton* (DFA) if its current state is always updated to a single state; otherwise, it is a *non-deterministic finite automaton* (NFA) if its currant state could refer to multiple possible states. The trade off between a DFA and an NFA representations for a RE is that the latter is more space-efficient while the former is more time-efficient for recognizing a matching string by checking a single path of state transitions. Let $|L(M)|$ denote the size of $L(M)$ and $|L_n(M)|$ denote the number of length-n strings in $L(M)$. Given a set $\mathcal{M}$ of finite automata, let $L(\mathcal{M})$ denote the language recognized by the automata in $\mathcal{M}$; i. e., $L(\mathcal{M}) = \bigcup_{M_i \in \mathcal{M}} L(M_i)$.

Key Results

The RE retrieval problem was first studied for a restricted class of REs in the context of content-based dissemination of XML documents using XPath-based subscriptions (e. g., [1,3,7]), where each XPath expression is processed in terms of a collection of path expressions. While the XPath language [6] allows rich patterns with tree structure to be specified, the path expressions that it supports lack the full expressive power of REs (e. g., XPath does not permit the RE operators *, $+$ and $\cdot$ to be arbitrarily nested in path expressions), and thus extending these XML-filtering techniques to handle general REs may not be straightforward. Further, all of the XPath-based methods are designed for indexing main-memory resident data. Another possible approach would be to coalesce the automata for all the REs into a single NFA, and then use this structure to determine the collection of matching REs. It is unclear, however, if the performance of such an approach would be superior to a simple sequential scan over the database of REs; furthermore, it is not easy to see how such a scheme could be adapted for disk-resident RE data sets.

The first disk-based data structure that can handle the storage and retrieval of REs in their full generality is the *RE-tree* [4,5]. Similar to the R-tree [8], an RE-tree is a dynamic, height-balanced, hierarchical index structure, where the leaf nodes contain data entries corresponding to the indexed REs, and the internal nodes contain "directory" entries that point to nodes at the next level of the index. Each leaf node entry is of the form (id, M), where id is the unique identifier of an RE r and M is a finite automaton representing r. Each internal node stores a collection of finite automata; and each node entry is of the form (M, ptr), where M is a finite automaton and ptr is a pointer to some node N (at the next level) such that the following *containment property* is satisfied: If $\mathcal{M}_N$ is the collection of automata contained in node N, then $L(\mathcal{M}_N) \subseteq L(M)$. The automaton M is referred to as the *bounding automaton* for $\mathcal{M}_N$. The containment property is key to improving the search performance of hierarchical index structures like RE-trees: if a query string w is not contained in $L(M)$, then it follows that $w \notin L(M_i)$ for all $M_i \in \mathcal{M}_N$. As a result, the entire subtree rooted at N can be pruned from the search space. Clearly, the closer $L(M)$ is to $L(\mathcal{M}_N)$, the more effective this search-space pruning will be.

In general, there are an infinite number of bounding automata for $\mathcal{M}_N$ with different degrees of precision from the least precise bounding automaton with $L(M) = \Sigma^*$ to the most precise bounding automaton, referred to as the *minimal bounding automaton*, with $L(M) = L(\mathcal{M}_N)$. Since the storage space for an automaton is dependent on its complexity (in terms of the number of its states and transitions), there is a space-precision tradeoff involved in the choice of a bounding automaton for each internal node entry. Thus, even though minimal bounding automata result in the best pruning due to their tightness, it may not be desirable (or even feasible) to always store minimal bounding automata in RE-trees since their space requirement can be too large (possibly exceeding the size of an index node), thus resulting in an index structure with a low fan-out. Therefore, to maintain a reasonable fan-out for RE-trees, a space constraint is imposed on the maximum number of states (denoted by α) permitted for each bounding automaton in internal RE-tree nodes. The automata stored in RE-tree nodes are, in general, NFAs with a minimum number of states. Also, for better space utilization, each individual RE-tree node is required to contain at least m entries. Thus, the RE-tree height is $O(\log_m(|S|))$.

RE-trees are conceptually similar to other hierarchical, spatial index structures, like the R-tree [8] that is designed for indexing a collection of multi-dimensional rectangles, where each internal entry is represented by a minimal bounding rectangle (MBR) that contains all the rectangles in the node pointed to by the entry. RE-tree search simply proceeds top-down along (possibly) multiple paths whose bounding automaton accepts the input string; RE-tree updates try to identify a "good" leaf node for insertion and can lead to node splits (or, node merges for deletions) that can propagate all the way up to the root. There is, however, a fundamental difference between the RE-tree and the R-tree in the indexed data types: regular languages typically represent *infinite* sets with no well-defined notion of spatial locality. This difference mandates the development of novel algorithmic solutions for the core RE-tree operations. To optimize for search performance, the core RE-tree operations are designed to keep each bounding automaton M in every internal node to be as "tight" as possible. Thus, if M is the bounding automaton for $\mathcal{M}_N$, then $L(M)$ should be as close to $L(\mathcal{M}_N)$ as possible.

There are three core operations that need to be addressed in the RE-tree context: (P1) selection of an optimal insertion node, (P2) computing an optimal node split, and (P3) computing an optimal bounding automaton. The goal of (P1) is to choose an insertion path for a new RE that leads to "minimal expansion" in the bounding automaton of each internal node of the insertion path. Thus, given the collection of automata $\mathcal{M}(N)$ in an internal index node N and a new automaton M, an optimal $M_i \in \mathcal{M}(N)$ needs to be chosen to insert M such that $|L(M_i) \cap L(M)|$ is maximum. The goal of (P2), which arises when splitting a set of REs during an RE-tree node-split, is to identify a partitioning that results in the minimal amount of "covered area" in terms of the languages of the resulting partitions. More formally, given the collection of automata $\mathcal{M} = \{M_1, M_2, \cdots, M_k\}$ in an overflowed index node, find the optimal partition of $\mathcal{M}$ into two disjoint subsets $\mathcal{M}_1$ and $\mathcal{M}_2$ such that $|\mathcal{M}_1| \geq m$, $|\mathcal{M}_2| \geq m$ and $|L(\mathcal{M}_1)| + |L(\mathcal{M}_2)|$ is minimum. The goal of (P3), which arises during insertions, node-splits, or node-merges, is to identify a bounding automaton for a set of REs that does not cover too much "dead space". Thus, given a collection of automata $\mathcal{M}$, the goal is to find the optimal bounding automaton M such that the number of states of M is no more than α, $L(\mathcal{M}) \subseteq L(M)$ and $|L(M)|$ is minimum.

The objective of the above three operations is to maximize the pruning during search by keeping bounding automata tight. In (P1), the optimal automaton M_i selected (within an internal node) to accommodate a newly inserted automaton M is to maximize $|L(M_i) \cap L(M)|$. The set of automata $\mathcal{M}$ are split into two tight clusters in (P2), while in (P3), the most precise automaton (with no more than α states) is computed to cover the set of automata in $\mathcal{M}$. Note that (P3) is unique to RE-trees, while both (P1) and (P2) have their equivalents in R-trees. The heuristics solutions [2,8] proposed for (P1) and (P2) in R-trees aim to minimize the number of visits to nodes that do not lead to any qualifying data entries. Although the minimal bounding automata in RE-trees (which correspond to regular languages) are very different from the MBRs in R-trees, the intuition behind minimizing the area of MBRs (total area or overlapping area) in R-trees should be effective for RE-trees as well. The counterpart for area in an RE-tree is $|L(M)|$, the size of the regular language for M. However, since a regular language is generally an infinite set, new measures need to be developed for the size of a regular language or for comparing the sizes of two regular languages.

One approach to compare the relative sizes of two regular languages is based on the following definition: for a pair of automata M_i and M_j, $L(M_i)$ is said to be larger than $L(M_j)$ if there exists a positive integer N such that for all $k \geq N$, $\sum_{l=1}^{k} |L_l(M_i)| \geq \sum_{l=1}^{k} |L_l(M_j)|$. Based on the above intuition, three increasingly sophisticated measures are proposed to capture the size of an infinite regular language. The *max-count measure* simply counts the number of strings in the language up to a certain size λ; i. e., $|L(M)| = \sum_{i=1}^{\lambda} |L_i(M)|$. This measure is useful for applications where the maximum length of all the REs to be indexed are known and is not too large so that λ can be set to some value slightly larger than the maximum length of the REs. A second more robust measure that is less sensitive to the λ parameter value is the *rate-of-growth measure* which is based on the intuition that a larger language grows at a faster rate than a smaller language. The size of a language is approximated by computing the rate of change of its size from one "window" of lengths to the next consecutive "window" of lengths: if λ is a length parameter that denote the start of the first window and θ is a window-size parameter, then $|L(M)| = \sum_{\lambda+\theta}^{\lambda+2\theta-1} |L_i(M)| / \sum_{\lambda}^{\lambda+\theta-1} |L_i(M)|$. As in the max-count measure, the parameters λ and θ should be chosen to be slightly greater than the number of states of M to ensure that strings involving a substantial portion of paths, cycles, and accepting states are counted in each window. However, there are cases where the rate-of-growth measure also fails to capture the "larger than" relationship between regular languages [4]. To address some of the shortcomings of the first two metrics, a third information-theoretic measure is proposed that is based on Rissanen's Minimum description length (MDL) principle [11]. The intuition is that if $L(M_i)$ is larger than $L(M_j)$, then the per-symbol-cost of an MDL-based encoding of a random string in $L(M_i)$ using M_i is very likely to be higher than that of a string in $L(M_j)$ using M_j, where the per-symbol-cost of encoding a string $w \in L(M)$ is the ratio of the cost of an MDL-based encoding of w using M to the length of w. More specifically, if $w = w_1.w_2.\cdots.w_n \in L(M)$ and $s_0, s_1, \ldots, s_n$ is the unique sequence of states visited by w in M, then the MDL-based encoding cost of w using M is given by $\sum_{i=0}^{n-1} \lceil \log_2(n_i) \rceil$, where each n_i denotes the number of transitions out of state s_i, and $\log_2(n_i)$ is the number of bits required to specify the transition out of state s_i. Thus, a reasonable measure for the size of a regular language $L(M)$ is the expected per-symbol-cost of an MDL-based encoding for a random sample of strings in $L(M)$.

To utilize the above metrics for measuring $L(M)$, one common operation needed is the computation of $|L_n(M)|$, the number of length-n strings in $L(M)$. While $|L_n(M)|$ can be efficiently computed when M is a DFA, the problem becomes #P-complete when M is an NFA [10]. Two approaches were proposed to approximate $|L_n(M)|$ when N is an NFA [10]. The first approach is an unbiased estimator for $|L_n(M)|$, which can be efficiently computed but can have a very large standard deviation. The second approach is a more accurate randomized algorithm for approximating $|L_n(M)|$ but it is not very useful in practice due to its high time complexity of $O(n^{\log(n)})$. A more practical approximation algorithm with a time complexity of $O(n^2|M|^2 \min\{|\Sigma|, |M|\})$ was proposed in [4].

The RE-tree operations (P1) and (P2) require frequent computations of $|L(M_i \cap M_j)|$ and $|L(M_i \cup M_j)|$ to be performed for pairs of automata M_i, M_j. These computations can adversely affect RE-tree performance since construction of the intersection and union automaton M can be expensive. Furthermore, since the final automaton M may have many more states than the two initial automata M_i and M_j, the cost of measuring $|L(M)|$ can be high. The performance of these computations can, however, be optimized by using sampling. Specifically, if the counts and samples for each $L(M_i)$ are available, then this information can be utilized to derive approximate counts and samples for $L(M_i \cap M_j)$ and $L(M_i \cup M_j)$ without incurring the overhead of constructing the automata $M_i \cap M_j$ and $M_i \cup M_j$ and counting their sizes. The sampling techniques used are based on the following results for approximating the sizes of and generating uniform samples of unions and intersections of arbitrary sets:

Theorem 1 (Chan, Garofalakis, Rastogi, [4]) *Let r_1 and r_2 be uniform random samples of sets S_1 and S_2, respectively.*

1. *$(|r_1 \cap S_2||S_1|)/|r_1|$ is an unbiased estimator of the size of $S_1 \cap S_2$.*
2. *$r_1 \cap S_2$ is a uniform random sample of $S_1 \cap S_2$ with size $|r_1 \cap S_2|$.*
3. *If the sets S_1 and S_2 are disjoint, then a uniform random sample of $S_1 \cup S_2$ can be computed in $O(|r_1| + |r_2|)$ time. If S_1 and S_2 are not disjoint, then an approximate uniform random sample of $S_1 \cup S_2$ can be computed with the same time complexity.*

Applications

The RE retrieval problem also arises in the context of both XML document classification, which identifies matching DTDs for XML documents, as well as BGP routing, which assigns appropriate priorities to BGP advertisements based on their matching routing-system sequences.

Experimental Results

Experimental results with synthetic data sets [5] clearly demonstrate that the RE-tree index is significantly more effective than performing a sequential search for matching REs, and in a number of cases, outperforms sequential search by up to an order of magnitude.

Recommended Reading

1. Altinel, M., Franklin, M.: Efficient filtering of XML documents for selective dissemination of information. In: *Proceedings of 26th International Conference on Very Large Data Bases*, Cairo, Egypt, pp. 53–64. Morgan Kaufmann, Missouri (2000)
2. Beckmann, N., Kriegel, H.-P., Schneider, R., Seeger, B.: The R*-Tree: An efficient and robust access method for points and rectangles. In: *Proceedings of the ACM International Conference on Management of Data*, Atlantic City, New Jersey, pp. 322–331. ACM Press, New York (1990)
3. Chan, C.-Y., Felber, P., Garofalakis, M., Rastogi, R.: Efficient filtering of XML documents with XPath expressions. In: *Proceedings of the 18th International Conference on Data Engineering*, San Jose, California, pp. 235–244. IEEE Computer Society, New Jersey (2002)
4. Chan, C.-Y., Garofalakis, M., Rastogi, R.: RE-Tree: An efficient index structure for regular expressions. In: *Proceedings of 28th International Conference on Very Large Data Bases*, Hong Kong, China, pp. 251–262. Morgan Kaufmann, Missouri (2002)

5. Chan, C.-Y., Garofalakis, M., Rastogi, R.: RE-Tree: An efficient index structure for regular expressions. *VLDB J.* **12**(2), 102–119 (2003)
6. Clark, J., DeRose, S.: XML Path Language (XPath) Version 1.0. W3C Recommendation, http://www.w3.org./TR/xpath, Accessed Nov 1999
7. Diao, Y., Fischer, P., Franklin, M., To, R.: YFilter: Efficient and scalable filtering of XML documents. In: *Proceedings of the 18th International Conference on Data Engineering*, San Jose, California, pp. 341–342. IEEE Computer Society, New Jersey (2002)
8. Guttman, A.: R-Trees: A dynamic index structure for spatial searching. In: *Proceedings of the ACM International Conference on Management of Data*, Boston, Massachusetts, pp. 47–57. ACM Press, New York (1984)
9. Hopcroft, J., Ullman, J.: *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Massachusetts (1979)
10. Kannan, S., Sweedyk, Z., Mahaney, S.: Counting and random generation of strings in regular languages. In: *Proceedings of the 6th ACM-SIAM Symposium on Discrete Algorithms*, San Francisco, California, pp. 551–557. ACM Press, New York (1995)
11. Rissanen, J.: Modeling by Shortest Data Description. Automatica **14**, 465–471 (1978)
12. Stewart, J.W.: *BGP4, Inter-Domain Routing in the Internet*. Addison Wesley, Massacuhsetts (1998)

Regular Expression Matching

2004; Navarro, Raffinot

LUCIAN ILIE
Department of Computer Science, University of Western Ontario, London, ON, Canada

Keywords and Synonyms

Automata-based searching

Problem Definition

Given a *text string* T of length n and a *regular expression* R, the **regular expression matching** problem **(REM)** is to find all text positions at which an occurrence of a string in $L(R)$ ends (see below for definitions).

For an alphabet Σ, a *regular expression* R over Σ consists of elements of $\Sigma \cup \{\varepsilon\}$ (ε denotes the empty string) and operators $\cdot$ (concatenation), | (union), and $*$ (iteration, that is, repeated concatenation); the set of strings $L(R)$ represented by R is defined accordingly; see [5]. It is important to distinguish two measures for the size of a regular expression: the *size*, m, which is the total number of characters from $\Sigma \cup \{\cdot, |, *\}$, and Σ*-size*, m_Σ, which counts only the characters in Σ. As an example, for $R = (\mathtt{A}|\mathtt{T})((\mathtt{C}|\mathtt{CG})*)$, the set $L(R)$ contains all strings that start with an A or a T followed by zero or more strings in the set {C, CG}; the size of R is $m = 8$ and the Σ-size is $m_\Sigma = 5$. Any regular expression can be processed in linear time so that $m = \mathcal{O}(m_\Sigma)$ (with a small constant); the difference becomes important when the two sizes appear as exponents.

Key Results

Finite Automata

The classical solutions for the REM problem involve finite automata which are directed graphs with the edges labeled by symbols from $\Sigma \cup \{\varepsilon\}$; their nodes are called states; see [5] for details. Unrestricted automata are called *nondeterministic finite automata (NFA)*. *Deterministic finite automata (DFA)* have no ε-labels and require that no two outgoing edges of the same state have the same label. Regular expressions and DFAs are equivalent, that is, the sets of strings represented are the same, as shown by Kleene [8]. There are two classical ways of computing an NFA from a regular expression. Thompson's construction [14], builds an NFA with up to $2m$ states and up to $4m$ edges whereas Glushkov–McNaughton–Yamada's automaton [3,9] has the minimum number of states, $m_\Sigma + 1$, and $\mathcal{O}(m_\Sigma^2)$ edges; see Fig. 1. Any NFA can be converted into an equivalent DFA by the *subset construction*: each subset of the set of states of the NFA becomes a state of the DFA. The problem is that the DFA can have exponentially more states than the NFA. For instance, the regular expression $((\mathtt{a}|\mathtt{b})*)\mathtt{a}(\mathtt{a}|\mathtt{b})(\mathtt{a}|\mathtt{b})\ldots(\mathtt{a}|\mathtt{b})$, with k occurrences of the $(\mathtt{a}|\mathtt{b})$ term, has a $(k+2)$-state NFA but requires $\Omega(2^k)$ states in any equivalent DFA.

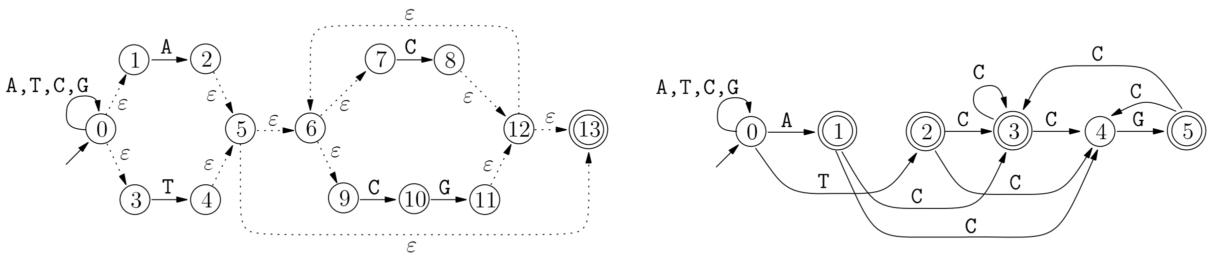
Classical Solutions

A regular expression is first converted into an NFA or DFA which is then simulated on the text. In order to be able to search for a match starting anywhere in the text, a loop labeled by all elements of Σ is added to the initial state; see Fig. 1.

Searching with an NFA requires linear space but many states can be active at the same time and to update them all one needs, for Thompson's NFA, $\mathcal{O}(m)$ time for each letter of the text; this gives Theorem 1. On the other hand, DFAs allow searching time that is linear in n but require more space for the automaton. Theorem 2 uses the DFA obtained from the Glushkov–McNaughton–Yamada's NFA.

Theorem 1 (Thompson [14]) *The REM problem can be solved with an NFA in* $\mathcal{O}(mn)$ *time and* $\mathcal{O}(m)$ *space.*

Theorem 2 (Kleene [8]) *The REM problem can be solved with a DFA in* $\mathcal{O}(n + 2^{m_\Sigma})$ *time and* $\mathcal{O}(2^{m_\Sigma})$ *space.*



Regular Expression Matching, Figure 1
Thompson's NFA (*left*) and Glushkov–McNaughton–Yamada's NFA (*right*) for the regular expression `(A|T)((C|CG)*)`; the initial loops labeled `A,T,C,G` are not part of the construction, they are needed for REM

Lazy Construction and Modules

One heuristic to alleviate the exponential increase in the size of DFA is to build only the states reached while scanning the text, as implemented in *Gnu Grep*. Still, the space needed for the DFA remains a problem. A four-Russians approach was presented by Myers [10] where a tradeoff between the NFA and DFA approaches is proposed. The syntax tree of the regular expression is divided into modules which are implemented as DFAs and are thereafter treated as leaf nodes in the syntax tree. The process continues until a single module is obtained.

Theorem 3 (Myers [10]) *The REM problem can be solved in $\mathcal{O}(mn/\log n)$ time and $\mathcal{O}(mn/\log n)$ space.*

Bit-Parallelism

The simulation of the above mentioned modules is done by encoding all states as bits of a single computer word (called *bit mask*) so that all can be updated in a single operation. The method can be used without modules, to simulate directly an NFA as done in [17] and implemented in the *Agrep* software [16]. Note that, in fact, the DFA is also simulated: a whole bit mask corresponds to a subset of states of the NFA, that is, one state of the DFA.

The bit-implementation of Wu and Manber [17] uses the property of Thompson's automaton that all Σ-labeled edges connect consecutive states, that is, they carry a bit 1 from position i to position $i+1$. This makes it easy to deal with the Σ-labeled edges but the ε-labeled ones are more difficult. A table of size linear in the number of states of the DFA needs to be precomputed to account for the ε-closures (set of states reachable from a given state by ε-paths).

Note that in Theorems 1, 2, and 3 the space complexity is given in words. In Theorems 4 and 5 below, for a more practical analysis, the space is given in bits and the alphabet size is also taken into consideration. For comparison, the space in Theorem 2, given in bits, is $\mathcal{O}(|\Sigma| m_\Sigma 2^{m_\Sigma})$.

Theorem 4 (Wu and Manber [17]) *Thompson's automaton can be implemented using $2m(2^{2m+1} + |\Sigma|)$ bits.*

Glushkov–McNaughton–Yamada's automaton has different structural properties. First, it is ε-free, that is, there are no ε-labels on edges. Second, all edges incoming to a given state are labeled the same. These properties are exploited by Navarro and Raffinot [13] to construct a bit-parallel implementation that requires less space. The results is a simple algorithm for regular expression searching which uses less space and usually performs faster than any existing algorithm.

Theorem 5 (Navarro and Raffinot [13]) *Glushkov–McNaughton–Yamada's automaton can be implemented using $(m_\Sigma + 1)(2^{m_\Sigma+1} + |\Sigma|)$ bits.*

All algorithms in this category run in $\mathcal{O}(n)$ time but smaller DFA representation implies more locality of reference and thus faster algorithms in practice. An improvement of any algorithm using Glushkov–McNaughton–Yamada's automaton can be done by reducing first the automaton by merging some of its states, as done by Ilie et al. [6]. The reduction can be performed in such a way that all useful properties of the automaton are preserved. The search becomes faster due to the reduction in size.

Filtration

The above approaches examine every character in the text. In [15] a multipattern search algorithm is used to search for strings that must appear inside any occurrence of the regular expression. Another technique is used in *Gnu Grep*; it extracts the longest string that must appear in any match (it can be used only when such a string exists). In [13], bit-parallel techniques are combined with a reverse factor search approach to obtain a very fast character skipping algorithm for regular expression searching.

Related Problems

Regular expressions with *backreference* have a feature that helps remembering what was matched to be used later; the matching problem becomes NP-complete; see [1]. *Extended* regular expressions involve adding two extra operators, intersection and complement, which do not change the expressive power. The corresponding matching problem can be solved in $\mathcal{O}((n+m)^4)$ time using dynamic programming, see [5, Exercise 3.23].

Concerning finite automata construction, recall that Thompson's NFA has $\mathcal{O}(m)$ edges whereas the ε-free Glushkov–McNaughton–Yamada's NFA can have a quadratic number of edges. It has been shown in [2] that one can always build an ε-free NFA with $\mathcal{O}(m \log m)$ edges (for fixed alphabets). However, it is the number of states which is more important in the searching algorithms.

Applications

Regular expression matching is a powerful tool in text-based applications, such as text retrieval and text editing, and in computational biology to find various motifs in DNA and protein sequences. See [4] for more details.

Open Problems

The most important theoretical problem is whether linear time and linear space can be achieved simultaneously. Characterizing the regular expressions that can be searched for using a linear-size equivalent DFA is also of interest. The expressions consisting of a single string are included here – the algorithm of Knuth, Morris, and Pratt is based on this. Also, it is not clear how much an NFA can be efficiently reduced (as done by [6]); the problem of finding a minimal NFA is PSPACE-complete, see [7]. Finally, for testing, it is not clear how to define random regular expressions.

Experimental Results

A disadvantage of the bit-parallel technique compared with the classical implementation of a DFA is that the former builds all possible subsets of states whereas the latter builds only the states that can be reached from the initial one (the other ones are useless). On the other hand, bit-parallel algorithms are simpler to code, more flexible (they allow also approximate matching), and there are techniques for reducing the space required. Among the bit-parallel versions, Glushkov–McNaughton–Yamada-based algorithms are better than Thompson-based ones. Modules obtain essentially the same complexity as bit-parallel ones but are more complicated to implement and slower in practice. As the number of computer words increases, bit-parallel algorithms slow down and modules may become attractive. Note also that technological progress has more impact on the bit-parallel algorithms, as opposed to classical ones, since the former depend very much on the machine word size. For details on comparison among various algorithms (including filtration based) see [12]; more recent comparisons are in [13], including the fastest algorithms to date.

URL to Code

Many text editors and programming languages include regular expression search features. They are, as well, among the tools used in protein databases, such as PROSITE and SWISS-PROT, which can be found at http://www.expasy.org/. The package *agrep* [17] can be downloaded from http://webglimpse.net/download.html and *nrgrep* [11] from http://www.dcc.uchile.cl/gnavarro/software.

Cross References

► Approximate Regular Expression Matching is a more general problem where errors are allowed.

Recommended Reading

1. Aho, A.: Algorithms for Finding Patterns in Strings. In: van Leewen, J. (ed.) Handbook of Theoretical Computer Science, vol. A: Algorithms and Complexity, pp. 255–300. Elsevier Science, Amsterdam and MIT Press, Cambridge (1990)
2. Geffert, V.: Translation of binary regular expressions into nondeterministic ε-free automata with $\mathcal{O}(n \log n)$ transitions. J. Comput. Syst. Sci. **66**(3), 451–472 (2003)
3. Glushkov, V.M.: The abstract theory of automata. Russ. Math. Surv. **16**, 1–53 (1961)
4. Gusfield, D.: Algorithms on Strings, Trees and Sequences. Cambridge University Press, Cambridge (1997)
5. Hopcroft, J., Ullman, J.: Introduction to Automata, Languages, and Computation. Addison-Wesley, Reading, MA (1979)
6. Ilie, L., Navarro, G., Yu, S.: On NFA reductions. In: Karhumäki, J. et al. (eds.) Theory is Forever. Lect. Notes Comput. Sci. **3113**, 112–124 (2004)
7. Jiang, T., Ravikumar, B.: Minimal NFA problems are hard. SIAM J. Comput. **22**(6), 1117–1141 (1993)
8. Kleene, S.C.: Representation of events in nerve sets. In: Shannon, C.E., McCarthy, J. (eds.) Automata Studies, pp. 3–40. Princeton Univ. Press, Princeton (1956)
9. McNaughton, R., Yamada, H.: Regular expressions and state graphs for automata. IRE Trans. Elect. Comput. **9**(1), 39–47 (1960)
10. Myers, E.: A four Russians algorithm for regular expression pattern matching. J. ACM **39**(2), 430–448 (1992)
11. Navarro, G.: Nr-grep: a fast and flexible pattern matching tool. Softw. Pr. Exp. **31**, 1265–1312 (2001)

12. Navarro, G., Raffinot, M.: Flexible Pattern Matching in Strings – Practical on-line search algorithms for texts and biological sequences. Cambridge University Press, Cambridge (2002)
13. Navarro, G., Raffinot, M.: New techniques for regular expression searching. Algorithmica **41**(2), 89–116 (2004)
14. Thompson, K.: Regular expression search algorithm. Commun. ACM **11**(6), 419–422 (1968)
15. Watson, B.: Taxonomies and Toolkits of Regular Language Algorithms, Ph. D. Dissertation, Eindhoven University of Technology, The Netherlands (1995)
16. Wu, S., Manber, U.: Agrep – a fast approximate patter-matching tool. In: Proceedings of the USENIX Technical Conf., pp. 153–162 (1992)
17. Wu, S., Manber, U.: Fast text searching allowing errors. Commun. ACM **35**(10), 83–91 (1992)

Reinforcement Learning

1992; Watkins

Eyal Even-Dar
Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA, USA

Keywords and Synonyms

Neuro dynamic programming

Problem Definition

Many sequential decision problems ranging from dynamic resource allocation to robotics can be formulated in terms of stochastic control and solved by methods of Reinforcement learning. Therefore, Reinforcement learning (a.k.a Neuro Dynamic Programming) has become one of the major approaches to tackling real life problems.

In Reinforcement learning, an agent wanders in an unknown environment and tries to maximize its long term return by performing actions and receiving rewards. The most popular mathematical models to describe Reinforcement learning problems are the Markov Decision Process (MDP) and its generalization Partially Observable MDP. In contrast to supervised learning, in Reinforcement learning the agent is learning through interaction with the environment and thus influences the "future". One of the challenges that arises in such cases is the exploration-exploitation dilemma. The agent can choose either to exploit its current knowledge and perhaps not learn anything new or to explore and risk missing considerable gains.

While Reinforcement learning contains many problems, due to lack of space this entry focuses on the basic ones. For a detailed history of the development of Reinforcement learning, see [13] chapter 1, the focus of the entry is on Q-learning and Rmax.

Notation

Markov Decision Process: A Markov Decision Process (MDP) formalizes the following problem. An agent is in an environment, which is composed of different states. In each time step the agent performs an action and as a result observes a signal. The signal is composed from the reward to the agent and the state it reaches in the next time step. More formally the MDP is defined as follows,

Definition 1 A Markov Decision process (MDP) M is a 4-tuple (S, A, P, R), where S is a set of the states, A is a set of actions, $P^a_{s,s'}$ is the transition probability from state s to state s' when performing action $a \in A$ in state s, and $R(s,a)$ is the reward distribution when performing action a in state s.

A strategy for an MDP assigns, at each time t, for each state s a probability for performing action $a \in A$, given a history $F_{t-1} = \{s_1, a_1, r_1, \dots, s_{t-1}, a_{t-1}, r_{t-1}\}$ which includes the states, actions and rewards observed until time $t-1$. While executing a strategy π an agent performs at time t action a_t in state s_t and observe a reward r_t (distributed according to $R(s_t, a_t)$), and a next state s_{t+1} (distributed according to $P^{a_t}_{s_t,\cdot}$). The sequence of rewards is combined into a single value called the *return*. The agent's goal is to maximize the return. There are several natural ways to define the return.

- *Finite horizon:* The return of policy π for a given horizon H is $\sum_{t=0}^{H} r_t$.
- *Discounted return:* For a discount parameter $\gamma \in (0,1)$, the discounted return of policy π is $\sum_{t=0}^{\infty} \gamma^t r_t$.
- *Undiscounted return:* The return of policy π is $\lim_{t\to\infty} \frac{1}{t+1} \sum_{i=0}^{t} r_i$.

Due to to lack of space, only discounted return, which is the most popular approach mainly due to its mathematical simplicity, is considered. The value function for each state s, under policy π, is defined as $V^\pi(s) = E^\pi[\sum_{i=0}^{\infty} r_i \gamma^i]$, where the expectation is over a run of policy π starting at state s. The state-action value function for using action a in state s and then following π is defined as $Q^\pi(s,a) = R(s,a) + \gamma \sum_{s'} P^a_{s,s'} V^\pi(s')$.

There exists a stationary deterministic optimal policy, π^*, which maximizes the return from any start state [11]. This implies that for any policy π and any state s, $V^{\pi^*}(s) \geq V^\pi(s)$, and $\pi^*(s) = \operatorname{argmax}_a (Q^{\pi^*}(s,a))$. A policy π is ε-optimal if $\| V^{\pi^*} - V^\pi \|_\infty \leq \epsilon$.

Problems Formulation

The Reinforcement learning problems are divided into two categories, planning and learning.

Planning: Given an MDP in its tabular form compute the optimal policy. An MDP is given in its tabular form if the 4-tuple, (A, S, P, R) is given explicitly.

The standard methods for the planning problem in MDP are given below.

Value Iteration: The value iteration is defined as follows. Start with some initial value function, C_s and then iterate using the Bellman operator, $TV(s) = \max_a R(s, a) + \gamma \sum_{s' \in S} P^a_{s,s'} V(s')$.

$$V_0(s) = C_s$$
$$V_{t+1}(s) = TV_t(s) ,$$

This method relies on the fact that the Bellman operator is contracting. Therefore, the distance between the optimal value function and current value function contracts by a factor of γ with respect to max norm (L_∞) in each iteration.

Policy Iteration: This algorithm starts with initial policy π_0 and iterates over polices. The algorithm has two phases for each iteration. In the first phase, the *Value evaluation step*, a value function for π_t is calculated, by finding the fixed point of $T_{\pi_t} V_{\pi_t} = V_{\pi_t}$, where $T_{\pi_t} V = R(s, \pi_t(s)) + \gamma \sum_{s' \in S} P^{\pi_t(s)}_{s,s'} V(s')$. The second phase, *Policy Improvement step*, is taking the next policy, π_{t+1} as a greedy policy with respect to V_{π_t}. It is known that Policy iteration converges with fewer iterations than value iteration. In practice the convergence of Policy iteration is very fast.

Linear Programming: Formulates and solves an MDP as linear program (LP). The LP variables are $V_1,\ldots,V_n$, where $V_i = V(s_i)$. The definition is:

$$\begin{aligned} &\text{Variables:} && V_1, \ldots, V_n \\ &\text{Minimize:} && \sum_i V_i \\ &\text{Subject to:} && V_i \geq [R(s_i, a) + \gamma \sum_j P_{s_i,s_j}(a) V_j] \\ &&& \qquad \forall a \in A, s_i \in S. \end{aligned}$$

Learning: Given the states and action identities, learn an (almost)optimal policy through interaction with the environment. The methods are divided into two categories: model free learning and model based learning.

The widely used *Q*-learning [16] is a model free algorithm. This algorithm belongs to the class of temporal difference algorithms [12]. *Q*-learning is an off policy method, i. e. it does not depend on the underlying policy

```
Rmax
Set $K = \emptyset$;
if $s \in K$? then
    Execute $\hat{\pi}(s)$
else
    Execute a random action;
    if $s$ becomes known then
        $K = K \bigcup \{s\}$;
        Compute optimal policy, $\hat{\pi}$ for
        the modified empirical model
    end
end
```

Reinforcement Learning, Algorithm 1
A model based algorithm

and as immediately will be seen it depends on the trajectory and not on the policy generating the trajectory.

Q learning: The algorithm estimates the state-action value function (for discounted return) as follows:

$$Q_0(s, a) = 0$$
$$Q_{t+1}(s, a) = (1 - \alpha_t(s, a))Q_t(s, a) + \alpha_t(s, a)(r_t(s, a) + \gamma V_t(s'))$$

where s' is the state reached from state s when performing action a at time t, and $V_t(s) = \max_a Q_t(s, a)$. Assume that $\alpha_t(s', a') = 0$ if at time t action a' was not performed at state s'. A learning rate α_t is *well-behaved* if for every state action pair (s, a): (1) $\sum_{t=1}^\infty \alpha_t(s, a) = \infty$ and (2) $\sum_{t=1}^\infty \alpha_t^2(s, a) < \infty$. As will be seen this is necessary for the convergence of the algorithm.

The model based algorithms are very simple to describe; they simply build an empirical model and use any of the standard methods to find the optimal policy in the empirical (approximate) model. The main challenge in this methods is in balancing exploration and exploitation and having an appropriate stopping condition. Several algorithms give a nice solution for this [3,7]. A version of these algorithms appearing in [6] is described below.

On an intuitive level a state will become known when it was visited "enough" times and one can estimate with high probability its parameters with good accuracy. The modified empirical model is defined as follows. All states that are not in K are represented by a single absorbing state in which the reward is maximal (which causes exploration). The probability to move to the absorbing state from a state $s \in K$ is the empirical probability to move out of K from s and the probability to move between states in K is the empirical probability.

Sample complexity [6] measures how many samples an algorithm need in order to learn. Note that the sample complexity translates into the time needed for the agent to wander in the MDP.

Key Results

The first Theorem shows that the planning problem is easy as long as the MDP is given in its tabular form, and one can use the algorithms presented in the previous section.

Theorem 1 ([10]) *Given an MDP the planning problem is P-complete.*

The learning problem can be done also efficiently using the $R_{\max}$ algorithm as is shown below.

Theorem 2 ([3,7]) *$R_{\max}$ computes an ε-optimal policy from state s with probability at least $1-\delta$ with sample complexity polynomial in $|A|, |S|, \frac{1}{\epsilon}$ and $\log \frac{1}{\delta}$, where s is the state in which the algorithm halts. Also the algorithm's computational complexity is polynomial in $|A|$ and $|S|$.*

The fact that Q-learning converges in the limit to the optimal Q function (which guarantees that the greedy policy with respect to the Q function will be optimal) is now shown.

Theorem 3 ([17]) *If every state-action is visited infinitely often and the learning rate is well behaved then Q_t converges to Q^* with probability one.*

The last statement is regarding the convergence rate of Q-learning. This statement must take into consideration some properties of the underlying policy, and assume that this policy covers the entire state space in reasonable time. The next theorem shows that the convergence rate of Q-learning can vary according to the tuning of the algorithm parameters.

Theorem 4([4]) *Let L be the time needed for the underlying policy to visit every state action with probability 1/2. Let T be the time until $\|Q^* - Q_T\| \leq \epsilon$ with probability at least $1-\delta$ and $\#(s,a,t)$ be the number of times action a was performed at state s until time t. Then if $\alpha_t(s,a) = 1/\#(s,a,t)$, then T is polynomial in $L, \frac{1}{\epsilon}, \log\frac{1}{\delta}$ and exponential in $\frac{1}{1-\gamma}$. If $\alpha_t(s,a) = 1/\#(s,a,t)^\omega$ for $\omega \in (1/2,1)$, then T is polynomial $L, \frac{1}{\epsilon}, \log\frac{1}{\delta}$ and $\frac{1}{1-\gamma}$.*

Applications

The biggest successes of Reinforcement learning so far are mentioned here. For a list of Reinforcement learning successful applications see http://neuromancer.eecs.umich.edu/cgi-bin/twiki/view/Main/SuccessesOfRL.

Backgammon Tesauro [14] used Temporal difference learning combined with neural network to design a player who learned to play backgammon by playing itself, and result in one level with the world's top players.

Helicopter control Ng et al. [9] used inverse Reinforcement learning for autonomous helicopter flight.

Open Problems

While in this entry only MDPs given in their tabular form were discussed much of the research is dedicated to two major directions: large state space and partially observable environments.

In many real world applications, such as robotics, the agent cannot observe the state she is in and can only observes a signal which is correlated with it. In such scenarios the MDP framework is no longer suitable, and another model is in order. The most popular reinforcement learning for such environment is the Partially Observable MDP. Unfortunately, for POMDP even the planning problems are intractable (and not only for the optimal policy which is not stationary but even for the optimal stationary policy); the learning contains even more obstacles as the agent cannot repeat the same state twice with certainty and thus it is not obvious how she can learn. An interesting open problem is trying to characterize when a POMDP is "solvable" and when it is hard to solve according to some structure.

In most applications the assumption that the MDP can be be represented in its tabular form is not realistic and approximate methods are in order. Unfortunately not much theoretically is known under such conditions. Here are a few of the prominent directions to tackle large state space.

Function Approximation: The term function approximation is due to the fact that it takes examples from a desired function (e. g., a value function) and construct an approximation of the entire function. Function approximation is an instance of supervised learning, which is studied in machine learning and other fields. In contrast to the tabular representation, this time a parameter vector Θ represents the value function. The challenge will be to learn the optimal vector parameter in the sense of minimum square error, i. e.

$$\min_\Theta \sum_{s\in S} (V^\pi(s) - V(s,\Theta))^2 ,$$

where $V(s,\Theta)$ is the approximation function. One of the most important function approximations is the linear function approximation,

$$V_t(s,\Theta) = \sum_{i=1}^{T} \phi_s(i)\Theta_t(i) ,$$

where each state has a set of vector features, ϕ_s. A feature based function approximation was analyzed and demonstrated in [2,15]. The main goal here is designing algorithm which converge to almost optimal polices under realistic assumptions.

Factored Markov Decision Process: In a FMDP the set of states is described via a set of random variables $X = \{X_1, \dots, X_n\}$, where each X_i takes values in some finite domain $Dom(X_i)$. A state s defines a value $x_i \in Dom(X_i)$ for each variable X_i. The transition model is encoded using a dynamic Bayesian network. Although the representation is efficient, not only is finding an ε-optimal policy intractable [8], but it cannot be represented succinctly [1]. However, under few assumptions on the FMDP structure there exists algorithms such as [5] that have both theoretical guarantees and nice empirical results.

Cross References

▶ Attribute-Efficient Learning
▶ Learning Automata
▶ Learning Constant-Depth Circuits
▶ Mobile Agents and Exploration
▶ PAC Learning

Recommended Reading

1. Allender, E., Arora, S., Kearns, M., Moore, C., Russell, A.: Note on the representational incompatabilty of function approximation and factored dynamics. In: Advances in Neural Information Processing Systems 15, 2002
2. Bertsekas, D.P., Tsitsiklis, J. N.: Neuro-Dynamic Programming. Athena Scientific, Belmont (1996)
3. Brafman, R., Tennenholtz, M.: R-max – a general polynomial time algorithm for near optimal reinforcement learning. J. Mach. Learn. Res. **3**, 213–231 (2002)
4. Even-Dar, E., Mansour, Y.: Learning rates for Q-learning. J. Mach. Learn. Res. **5**, 1–25 (2003)
5. Guestrin, C., Koller, D., Parr, R., Venkataraman, S.: Efficient solution algorithms for factored mdps. J. Artif. Intell. Res. **19**, 399–468 (2003)
6. Kakade, S.: On the Sample Complexity of Reinforcement Learning. Ph. D. thesis, University College London (2003)
7. Kearns, M., Singh, S.: Near-optimal reinforcement learning in polynomial time. Mach. Learn. **49**(2–3), 209–232 (2002)
8. Lusena, C., Goldsmith, J., Mundhenk, M.: Nonapproximability results for partially observable markov decision processes. J. Artif. Intell. Res. **14**, 83–103 (2001)
9. Ng, A.Y., Coates, A., Diel, M., Ganapathi, V., Schulte, J., Tse, B., Berger, E., Liang, E.:Inverted autonomous helicopter flight via reinforcement learning. In: International Symposium on Experimental Robotics, 2004
10. Papadimitriu, C.H., Tsitsiklis, J.N.: The complexity of markov decision processes. In: Mathematics of Operations Research, 1987, pp. 441–450.
11. Puterman, M.: Markov Decision Processes. Wiley-Interscience, New York (1994)
12. Sutton, R.: Learning to predict by the methods of temporal differences. Mach. Learn. **3**, 9–44 (1988)
13. Sutton, R., Barto, A.: Reinforcement Learning. An Introduction. MIT Press, Cambridge (1998)
14. Tesauro, G.J.: TD-gammon, a self-teaching backgammon program, achieves a master-level play. Neural Comput. **6**, 215–219 (1996)
15. Tsitsiklis, J.N., Van Roy, B.: Feature-based methods for large scale dynamic programming. Mach. Learn. **22**, 59–94 (1996)
16. Watkins, C.: Learning from Delayed Rewards. Ph.D. thesis, Cambridge University (1989)
17. Watkins, C., Dyan, P.: Q-learning. Mach. Learn. **8**(3/4), 279–292 (1992)

Renaming

1990; Attiya, Bar-Noy, Dolev, Peleg, Reischuk

MAURICE HERLIHY
Department of Computer Science, Brown University, Providence, RI, USA

Keywords and Synonyms

Wait-free renaming

Problem Definition

Consider a system in which $n + 1$ processes $P_0, \dots, P_n$ communicate either by message-passing or by reading and writing a shared memory. Processes are *asynchronous*: there is no upper or lower bounds on their speeds, and up to t of them may fail undetectably by halting. In the *renaming task* proposed by Attiya, Bar-Noy, Dolev, Peleg, and Reischuk [1], each process is given a unique *input name* taken from a range $0, \dots, N$, and chooses a unique *output name* taken from a strictly smaller range $0, \dots, K$. To rule out trivial solutions, a process's decision function must depend only on input names, not its preassigned identifier (so that P_i cannot simply choose output name i). Attiya et al. showed that the task has no solution when $K = n$, but does have a solution when $K = N + t$. In 1993, Herlihy and Shavit [2] showed that the task has no solution when $K < N + t$.

Vertexes, simplexes, and complexes model decision tasks. (See the companion article entitled ▶ Topology Approach in Distributed Computing). A process's state at the start or end of a task is represented as a vertex $\vec{v}$ labeled with that process's identifier, and a value, either input or output: $\vec{v} = \langle P, v_i \rangle$. Two such vertexes are *compatible* if (1) they have distinct process identifiers, and (2) those process can be assigned those values together. For example, in the renaming task, input values are required to be distinct, so two input vertexes are compatible only if they are labeled with distinct process identifiers and distinct input values.

Figure 1 shows the output complex for the three-process renaming task using four names. Notice that the two edges marked A are identical, as are the two edges marked B. By identifying these edges, this task defines a simplicial complex that is topologically equivalent to a torus. Of course, after changing the number of processes or the number of names, this complex is no longer a torus.

Key Results

Theorem 1 *Let S^n be an n-simplex, and S^m a face of S^n. Let S be the complex consisting of all faces of S^m, and $\dot{S}$ the complex consisting of all proper faces of S^m (the boundary complex of S). If $\sigma(\dot{S})$ is a subdivision of $\dot{S}$, and $\phi\colon \sigma(\dot{S}) \to \mathcal{F}(S)$ a simplicial map, then there exists a subdivision $\tau(S)$ and a simplicial map $\psi\colon \tau(S) \to \mathcal{F}(S)$ such that $\tau(\dot{S}) = \sigma(\dot{S})$, and ϕ and ψ agree on $\sigma(\dot{S})$.*

Informally, any simplicial map of an m-sphere to $\mathcal{F}$ can be "filled in" to a simplicial map of the $(m+1)$-disk. A *span* for $\mathcal{F}(S^n)$ is a subdivision σ of the input simplex S^n together with a simplicial map $\phi\colon \sigma(S^n) \to \mathcal{F}(S^n)$ such that for every face S^m of S^n, $\phi\colon \sigma(S^m) \to \mathcal{F}(S^m)$. Spans are constructed one dimension at a time. For each $\vec{s} = \langle P_i, v_i \rangle \in S^n, \phi$ carries $\vec{s}$ to the solo execution by P_i with input $\vec{v}_i$. For each $S^1 = (\vec{s}_0, \vec{s}_1)$, Theorem 1 implies that $\phi(\vec{s}_0)$ and $\phi(\vec{s}_1)$ can be joined by a path in $\mathcal{F}(S^1)$. For each $S^2 = (\vec{s}_0, \vec{s}_1, \vec{s}_2)$, the inductively constructed spans define each face of the boundary complex $\phi\colon \sigma(S^1_{ij}) \to \mathcal{F}(S^1)_{ij}$, for $i, j \in \{0, 1, 2\}$. Theorem 1 implies that one can "fill in" this map, extending the subdivision from the boundary complex to the entire complex.

Theorem 2 *If a decision task has a protocol in asynchronous read/write memory, then each input simplex has a span.*

One can restrict attention to protocols that have the property that any process chooses the same name in a solo execution.

Definition 1 A protocol is *comparison-based* if the only operations a process can perform on processor identifiers is to test for equality and order; that is, given two P and Q, a process can test for $P = Q$, $P \le Q$, and $P \ge Q$, but cannot examine the structure of the identifiers in any more detail.

Lemma 3 *If a wait-free renaming protocol for K names exists, then a comparison-based protocol exists.*

Proof Attiya et al. [1] give a simple comparison-based wait-free renaming protocol that uses $2n+1$ output names. Use this algorithm to assign each process an *intermediate* name, and use that intermediate name as input to the K-name protocol. ☐

Comparison-based algorithms are *symmetric* on the boundary of the span. Let S^n be an input simplex, $\phi\colon \sigma(S^n) \to \mathcal{F}(S^n)$ a span, and $\mathcal{R}$ the output complex for $2n$ names. Composing the span map ϕ and the decision map δ yields a map $\sigma(S^n) \to \mathcal{R}$. This map can be simplified by replacing each output name by its parity, replacing the complex $\mathcal{R}$ with the binary n-sphere $\mathcal{B}^n$.

$$\mu\colon \sigma(S^n) \to \mathcal{B}^n\ . \tag{1}$$

Denote the simplex of $\mathcal{B}^n$ whose values are all zero by 0^n, and all one by 1^n.

Lemma 4 $\mu^{-1}(0^n) = \mu^{-1}(1^n) = \emptyset$.

Proof The range $0, \ldots, 2n-1$ does not contain $n+1$ distinct even names or $n+1$ distinct odd names. ☐

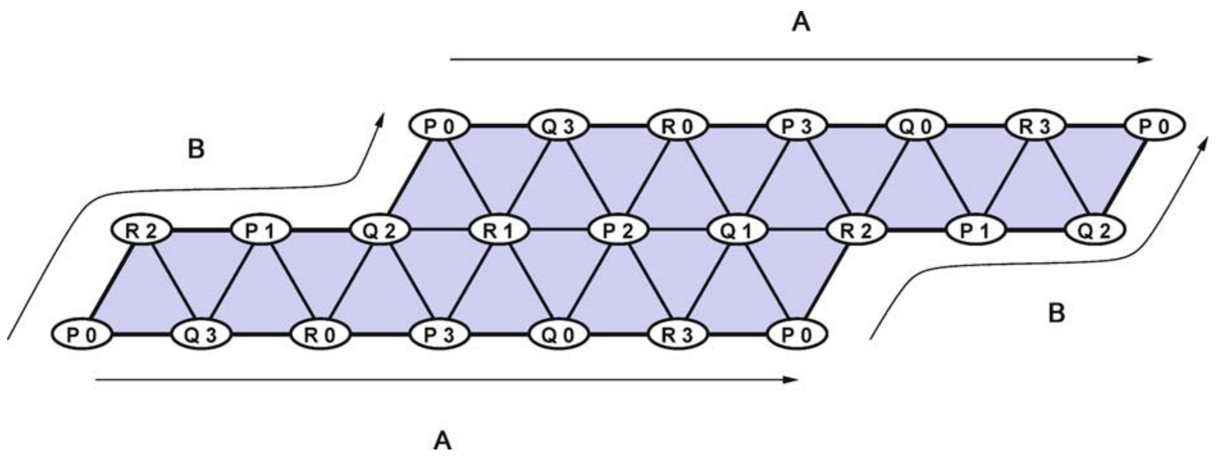
The *n-cylinder* C^n is the binary n-sphere without 0^n and 1^n. Informally, the rest of the argument proceeds by showing that the boundary of the span is "wrapped around" the hole in C^n a non-zero number of times.

The span $\sigma(S^n)$ (indeed any any subdivided n-simplex) is a (combinatorial) *manifold with boundary*: each $(n-1)$-simplex is a face of either one or two n-simplexes. If it is a face of two, then the simplex is an *internal simplex*, and otherwise it is a *boundary* simplex. An orientation of S^n induces an orientation on each n-simplex of $\sigma(S^n)$ so that each internal $(n-1)$-simplex inherits opposite orientations. Summing these oriented simplexes yields a chain, denoted $\sigma_*(S^n)$, such that

$$\partial\sigma_*(S^n) = \sum_{i=0}^{n} (-1)^i \sigma_*(\mathit{face}_i(S^n))\ .$$

The following is a standard result about the homology of spheres.

Theorem 5 *Let the chain 0^n be the simplex 0^n oriented like S^n. (1) For $0 < m < n$, any two m-cycles are homologous, and (2) every n-cycle C^n is homologous to $k \cdot \partial 0^n$, for some integer k. C^n is a boundary if and only if $k = 0$.*



Renaming, Figure 1
Output complex for 3-process renaming with 4 names

Let S^m be the face of S^n spanned by solo executions of $P_0, \ldots, P_m$. Let 0^m denote some m-simplex of C^n whose values are all zero. Which one will be clear from context.

Lemma 6 *For every proper face S^{m-1} of S^n, there is an m-chain $\alpha(S^{m-1})$ such that*

$$\mu_*(\sigma_*(S^m)) - 0^m - \sum_{i=0}^{m}(-1)^i \alpha(\mathit{face}_i(S^m))$$

is a cycle.

Proof By induction on m. When $m = 1$, $ids(S^1) = \{i, j\}$. 0^1 and $\mu_*(\sigma_*(S^1))$ are 1-chains with a common boundary $\langle P_i, 0\rangle - \langle P_j, 0\rangle$, so $\mu_*(\sigma_*(S^1)) - 0^1$ is a cycle, and $\alpha(\langle P_i, 0\rangle) = \emptyset$.

Assume the claim for $m, 1 \geq m < n-1$. By Theorem 5, every m-cycle is a boundary (for $m < n-1$), so there exists an $(m+1)$-chain $\alpha(S^m)$ such that

$$\mu_*(\sigma_*(S^m)) - 0^m - \sum_{i=0}^{m}(-1)^i \alpha(\mathit{face}_i(S^m)) = \partial\alpha(S^m)\,.$$

Taking the alternating sum over the faces of S^{m+1}, the $\alpha(\mathit{face}_i(S^m))$ cancel out, yielding

$$\mu_*(\partial\sigma_*(S^{m+1})) - \partial 0^{m+1} = \sum_{i=0}^{m+1}(-1)^i \partial\alpha(\mathit{face}_i(S^{m+1}))\,.$$

Rearranging terms yields

$$\partial\left(\mu_*(\sigma_*(S^{m+1})) - 0^{m+1} - \sum_{i=0}^{m+1}(-1)^i \alpha(\mathit{face}_i(S^{m+1}))\right) = 0\,,$$

implying that

$$\mu_*(\sigma_*(S^{m+1})) - 0^{m+1} - \sum_{i=0}^{m+1}(-1)^i \alpha(\mathit{face}_i(S^{m+1}))$$

is an $(m+1)$-cycle. □

Theorem 7 *There is no wait-free renaming protocol for $(n+1)$ processes using $2n$ output names.*

Proof Because

$$\mu_*(\sigma_*(S^{n-1})) - 0^{n-1} - \sum_{i=0}^{n}(-1)^i \alpha(\mathit{face}_i(S^{n-1}))$$

is a cycle, Theorem 5 implies that it is homologous to $k \cdot \partial 0^n$, for some integer k. Because μ is symmetric on the boundary of $\sigma(S^n)$, the alternating sum over the $(n-1)$-dimensional faces of S^n yields:

$$\mu_*(\partial\sigma_*(S^n)) - \partial 0^n \sim (n+1)k \cdot \partial 0^n$$

or

$$\mu_*(\partial\sigma_*(S^n)) \sim (1 + (n+1)k) \cdot \partial 0^n\,.$$

Since there is no value of k for which $(1 + (n+1)k)$ is zero, the cycle $\mu_*(\partial\sigma_*(S^n))$ is not a boundary, a contradiction. □

Applications

The renaming problem is a key tool for understanding the power of various asynchronous models of computation.

Open Problems

Characterizing the full power of the topological approach to proving lower bounds remains an open problem.

Cross References

▶ Asynchronous Consensus Impossibility
▶ Set Agreement
▶ Topology Approach in Distributed Computing

Recommended Reading

1. Attiya, H., Bar-Noy, A., Dolev, D., Peleg, D., Reischuk, R.: Renaming in an asynchronous environment. J. ACM **37**(3), 524–548 (1990)
2. Herlihy, M.P., Shavit, N.: The asynchronous computability theorem for *t*-resilient tasks. In: Proceedings 25th Annual ACM Symposium on Theory of Computing, 1993, pp. 111–120

Response Time

▶ Minimum Flow Time
▶ Shortest Elapsed Time First Scheduling

Reversal Distance

▶ Sorting Signed Permutations by Reversal (Reversal Distance)

RNA Secondary Structure Boltzmann Distribution

2005; Miklós, Meyer, Nagy

RUNE B. LYNGSØ
Department of Statistics, Oxford University, Oxford, UK

Keywords and Synonyms

Full partition function

Problem Definition

This problem is concerned with computing features of the Boltzmann distribution over RNA secondary structures in the context of the standard Gibbs free energy model used for RNA Secondary Structure Prediction by Minimum Free Energy (cf. corresponding entry). Thermodynamics state that for a system with configuration space Ω and free energy given by $E\colon\ \Omega \mapsto \mathbf{R}$, the probability of the system being in state $\omega \in \Omega$ is proportional to $\mathrm{e}^{-E(\omega)/RT}$ where R is the universal gas constant and T the absolute temperature of the system. The normalizing factor

$$Z = \sum_{\omega\in\Omega} \mathrm{e}^{-E(\omega)/RT} \qquad (1)$$

is called the *full partition function* of the system.

Over the past several decades, a model approximating the free energy of a structured RNA molecule by independent contributions of its secondary structure components has been developed and refined. The main purpose of this work has been to assess the stability of individual secondary structures. However, it immediately translates into a distribution over all secondary structures. Early work focused on computing the pairing probability for all pairs of bases, i. e. the sum of the probabilities of all secondary structures containing that base pair. Recent work has extended methods to compute probabilities of base pairing probabilities for RNA heterodimers [2], i. e. interacting RNA molecules, and expectation, variance and higher moments of the Boltzmann distribution.

Notation

Let $s \in \{A, C, G, U\}^*$ denote the sequence of bases of an RNA molecule. Use $X \cdot Y$ where $X, Y \in \{A, C, G, U\}$ to denote a base pair between bases of type X and Y, and $i \cdot j$ where $1 \le i < j \le |s|$ to denote a base pair between bases $s[i]$ and $s[j]$.

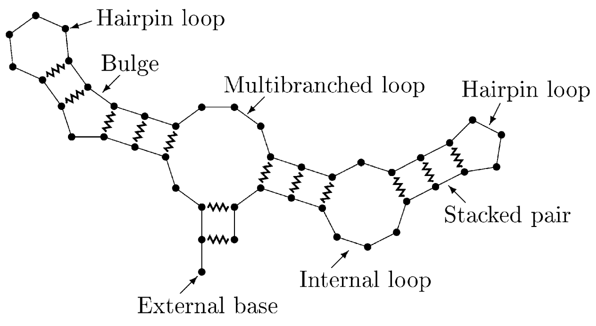
Definition 1 (RNA Secondary Structure) A secondary structure for an RNA sequence s is a set of base pairs $S = \{i \cdot j \mid 1 \le i < j \le |s| \wedge i < j - 3\}$. For $i \cdot j, i' \cdot j' \in S$ with $i \cdot j \neq i' \cdot j'$

- $\{i, j\} \cap \{i', j'\} = \emptyset$ (each base pairs with at most one other base)
- $\{s[i], s[j]\} \in \{\{A, U\}, \{C, G\}, \{G, U\}\}$ (only Watson-Crick and G, U wobble base pairs)
- $i < i' < j \Rightarrow j' < j$ (base pairs are either nested or juxtaposed but not overlapping)

The second requirement, that only canonical base pairs are allowed, is standard but not consequential in solutions to the problem. The third requirement states that the structure does not contain pseudoknots. This restriction is crucial for the results listed in this entry.

Energy Model

The model of Gibbs free energy applied, usually referred to as the nearest-neighbor model, was originally proposed by Tinoco et al. [10,11]. It approximates the free energy by postulating that the energy of the full three dimensional



RNA Secondary Structure Boltzmann Distribution, Figure 1
A hypothetical RNA structure illustrating the different loop types. Bases are represented by *circles*, the RNA backbone by *straight lines*, and base pairs by *zigzagged lines*

structure only depends on the secondary structure, and that this in turn can be broken into a sum of independent contributions from each loop in the secondary structure.

Definition 2 (Loops) For $i \cdot j \in S$, base k is *accessible* from $i \cdot j$ iff $i < k < j$ and $\neg\exists i' \cdot j' \in S : i < i' < k < j' < j$. The *loop closed by* $i \cdot j$, $\ell_{i \cdot j}$, consists of $i \cdot j$ and all the bases accessible from $i \cdot j$. If $i' \cdot j' \in S$ and i' and j' are accessible from $i \cdot j$, then $i' \cdot j'$ is an interior base pair in the loop closed by $i \cdot j$.

Loops are classified by the number of interior base pairs they contain:

- hairpin loops have no interior base pairs
- stacked pairs, bulges, and internal loops have one interior base pair that is separated from the closing base pair on neither side, on one side, or on both sides, respectively
- multibranched loops have two or more interior base pairs

Bases not accessible from any base pair are called external. This is illustrated in Fig. 1. The free energy of structure S is

$$\Delta G(S) = \sum_{i \cdot j \in S} \Delta G(\ell_{i \cdot j}) \tag{2}$$

where $\Delta G(\ell_{i \cdot j})$ is the free energy contribution from the loop closed by $i \cdot j$. The contribution of S to the full partition function is

$$\mathrm{e}^{-\Delta G(S)/RT} = \mathrm{e}^{-\sum_{i \cdot j \in S} \Delta G(\ell_{i \cdot j})/RT} = \prod_{\ell_{i \cdot j} \in S} \mathrm{e}^{-\Delta G(\ell_{i \cdot j})/RT} . \tag{3}$$

Problem 1 (RNA Secondary Structure Distribution)
INPUT: *RNA sequence s, absolute temperature T and specification of ΔG at T for all loops.*
OUTPUT: $\sum_S \mathrm{e}^{-\Delta G(S)/RT}$, *where the sum is over all secondary structures for s.*

Key Results

Solutions are based on recursions similar to those for RNA Secondary Structure Prediction by Minimum Free Energy, replacing sum and minimization with multiplication and sum (or more generally with a *merge function* and a *choice function* [8]). The key difference is that recursions are required to be non-redundant, i. e. any particular secondary structure only contributes through one path through the recursions.

Theorem 1 *Using the standard thermodynamic model for RNA secondary structures, the partition function can be computed in time $O(|s|^3)$ and space $O(|s|^2)$. Moreover, the computation can build data structures that allow O(1) queries of the pairing probability of $i \cdot j$ for any $1 \le i < j \le |s|$ [5,6,7].*

Theorem 2 *Using the standard thermodynamic model for RNA secondary structures, the expectation and variance of free energy over the Boltzmann distribution can be computed in time $O(|s|^3)$ and space $O(|s|^2)$. More generally, the kth moment*

$$E_{\text{Boltzmann}}[\Delta G] = 1/Z \sum_S \mathrm{e}^{-\Delta G(S)/RT} \Delta G^k(S) , \tag{4}$$

where $Z = \sum_S \mathrm{e}^{-\Delta G(S)/RT}$ is the full partition function and the sums are over all secondary structures for s, can be computed in time $O(k^2|s|^3)$ and space $O(k|s|^2)$ [8].

In Theorem 2 the free energy does not hold a special place. The theorem holds for any function Φ defined by an independent contribution from each loop,

$$\Phi(S) = \sum_{i \cdot j \in S} \phi\left(\ell_{i \cdot j}\right) , \tag{5}$$

provided each loop contribution can be handled with the same efficiency as the free energy contributions. Hence, moments over the Boltzmann distribution of e. g. number of base pairs, unpaired bases, or loops can also be efficiently computed by applying appropriately chosen indicator functions.

Applications

The original use of partition function computations was for discriminating between well defined and less well defined regions of a secondary structure. Minimum free energy predictions will always return a structure. Base pairing probabilities help identify regions where the prediction is uncertain, either due to the approximations of the model or that the real structure indeed does fluctuate between several low energy alternatives. Moments of Boltzmann distributions are used in identifying how biological RNA molecules deviates from random RNA sequences.

The data structures computed in Theorem 1 can also be used to efficiently sample secondary structures from the Boltzmann distribution. This has been used for probabilistic methods for secondary structure prediction, where the centroid of the most likely cluster of sampled structures is returned rather than the most likely, i. e. minimum free energy, structure [3]. This approach better accounts for the entropic effects of large neighborhoods of structurally and energetically very similar structures. As a simple illustration of this effect, consider twice flipping a coin with probability $p > 0.5$ for heads. The probability p^2 of heads in both flips is larger than the probability $p(1 - p)$ of heads followed by tails or tails followed by heads (which again is larger than the probability $(1 - p)^2$ of tails in both flips). However, if the order of the flips is ignored the probability of one heads and one tails is $2p(1 - p)$. The probability of two heads remains p^2 which is smaller than $2p(1 - p)$ when $p < \frac{2}{3}$. Similarly a large set of structures with fairly low free energy may be more likely, when viewed as a set, than a small set of structures with very low free energy.

Open Problems

As for RNA Secondary Structure Prediction by Minimum Free Energy, improvements in time and space complexity are always relevant. This may be more difficult for computing distributions, as the more efficient dynamic programming techniques of [9] cannot be applied. In the context of genome scans, the fact that the start and end positions of encoded RNA molecule is unknown has recently been considered [1].

Also the problem of including structures with pseudoknots, i. e. structures violating the last requirement in Def. 1, in the configuration space is an active area of research. It can be expected that all the methods of Theorems 3 through 6 in the entry on RNA Secondary Structure Prediction Including Pseudoknots can be modified to computation of distributions without affecting complexities. This may require some further bookkeeping to ensure non-redundancy of recursions, and only in [4] has this actively been considered.

Though the moments of functions that are defined as sums over independent loop contributions can be computed efficiently, it is unknown whether the same holds for functions with more complex definitions. One such function that has traditionally been used for statistics on RNA secondary structure [12] is the *order* of a secondary structure which refers to the nesting depth of multibranched loops.

URL to Code

Software for partition function computation and a range of related problems is available from www.bioinfo.rpi.edu/applications/hybrid/download.php and www.tbi.univie.ac.at/~ivo/RNA/. Software including a restricted class of structures with pseudoknots [4] is available at www.nupack.org.

Cross References

► RNA Secondary Structure Prediction Including Pseudoknots
► RNA Secondary Structure Prediction by Minimum Free Energy

Recommended Reading

1. Bernhart, S., Hofacker, I.L., Stadler, P.: Local RNA base pairing probabilities in large sequences. Bioinformatics **22**, 614–615 (2006)
2. Bernhart, S.H., Tafer, H., Mückstein, U., Flamm, C., Stadler, P.F., Hofacker, I.L.: Partition function and base pairing probabilities of RNA heterodimers. Algorithms Mol. Biol. **1**, 3 (2006)
3. Ding, Y., Chan, C.Y., Lawrence, C.E.: RNA secondary structure prediction by centroids in a Boltzmann weighted ensemble. RNA **11**, 1157–1166 (2005)
4. Dirks, R.M., Pierce, N.A.: A partition function algorithm for nucleic acid secondary structure including pseudoknots. J. Comput. Chem. **24**, 1664–1677 (2003)
5. Hofacker, I.L., Stadler, P.F.: Memory efficient folding algorithms for circular RNA secondary structures. Bioinformatics **22**, 1172–1176 (2006)
6. Lyngsø, R.B., Zuker, M., Pedersen, C.N.S.: Fast evaluation of internal loops in RNA secondary structure prediction. Bioinformatics **15**, 440–445 (1999)
7. McCaskill, J.S.: The equilibrium partition function and base pair binding probabilities for RNA secondary structure. Biopolymers **29**, 1105–1119 (1990)
8. Miklós, I., Meyer, I.M., Nagy, B.: Moments of the Boltzmann distribution for RNA secondary structures. Bull. Math. Biol. **67**, 1031–1047 (2005)
9. Ogurtsov, A.Y., Shabalina, S.A., Kondrashov, A.S., Roytberg, M.A.: Analysis of internal loops within the RNA secondary structure in almost quadratic time. Bioinformatics **22**, 1317–1324 (2006)
10. Tinoco, I., Borer, P.N., Dengler, B., Levine, M.D., Uhlenbeck, O.C., Crothers, D.M., Gralla, J.: Improved estimation of secondary structure in ribonucleic acids. Nature New Biol. **246**, 40–41 (1973)
11. Tinoco, I., Uhlenbeck, O.C., Levine, M.D.: Estimation of secondary structure in ribonucleic acids. Nature **230**, 362–367 (1971)
12. Waterman, M.S.: Secondary structure of single-stranded nucleic acids. Adv. Math. Suppl. Stud. **1**, 167–212 (1978)

RNA Secondary Structure Prediction Including Pseudoknots
2004; Lyngsø

RUNE B. LYNGSØ
Department of Statistics, Oxford University, Oxford, UK

Keywords and Synonyms

Abbreviated as *Pseudoknot Prediction*

Problem Definition

This problem is concerned with predicting the set of base pairs formed in the native structure of an RNA molecule, including overlapping base pairs also known as pseudoknots. Standard approaches to RNA secondary structure prediction only allow sets of base pairs that are hierarchically nested. Though few known real structures require the removal of more than a small percentage of their base pairs to meet this criteria, a significant percentage of known real structures contain at least a few base pairs overlapping other base pairs. Pseudoknot substructures are known to be crucial for biological function in several contexts. One of the more complex known pseudoknot structures is illustrated in Fig. 1

Notation

Let $s \in \{A, C, G, U\}^*$ denote the sequence of bases of an RNA molecule. Use $X \cdot Y$ where $X, Y \in \{A, C, G, U\}$ to denote a base pair between bases of type X and Y, and $i \cdot j$ where $1 \leq i < j \leq |s|$ to denote a base pair between bases $s[i]$ and $s[j]$.

Definition 1 (RNA Secondary Structure) A secondary structure for an RNA sequence s is a set of base pairs $S = \{i \cdot j \mid 1 \leq i < j \leq |s| \wedge i < j - 3\}$. For $i \cdot j, i' \cdot j' \in S$ with $i \cdot j \neq i' \cdot j'$

- $\{i, j\} \cap \{i', j'\} = \emptyset$ (each base pairs with at most one other base)
- $\{s[i], s[j]\} \in \{\{A, U\}, \{C, G\}, \{G, U\}\}$ (only Watson-Crick and G, U wobble base pairs)

The second requirement, that only canonical base pairs are allowed, is standard but not consequential in solutions to the problem.

Scoring Schemes

Structures are usually assessed by extending the model of Gibbs free energy used for RNA Secondary Structure Prediction by Minimum Free Energy (cf. corresponding entry) with *ad hoc* extrapolation of multibranched loop energies to pseudoknot substructures [11], or by summing independent contributions e. g. obtained from base pair restricted minimum free energy structures from each base pair [13]. To investigate the complexity of pseudoknot prediction the following three simple scoring schemes will also be considered:

Number of Base Pairs,

$$\#BP(S) = |S|$$

Number of Stacking Base Pairs

$$\#SBP(S) = |\{i \cdot j \in S \mid i+1 \cdot j-1 \in S \vee i-1 \cdot j+1 \in S\}|$$

Number of Base Pair Stackings

$$\#BPS(S) = |\{i \cdot j \in S \mid i+1 \cdot j-1 \in S\}|$$

These scoring schemes are inspired by the fact that stacked pairs are essentially the only loops having a stabilizing contribution in the Gibbs free energy model.

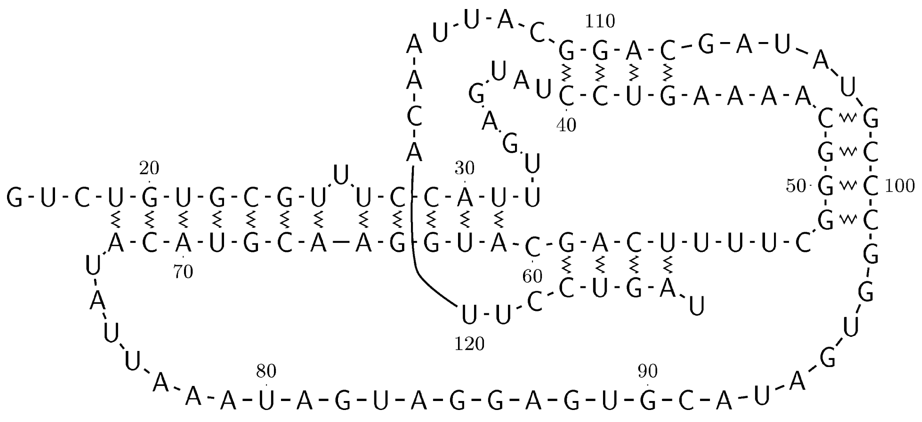
Problem 1 (Pseudoknot Prediction)
INPUT: *RNA sequence s and an appropriately specified scoring scheme.*
OUTPUT: *A secondary structure S for s that is optimal under the scoring scheme specified.*

Key Results

Theorem 1 *The complexities of pseudoknot prediction under the three simplified scoring schemes can be classified as follows, where Σ denotes the alphabet.*

	Fixed alphabet	*Unbounded alphabet*
#BP [13]	*Time* $O\left(\lvert s\rvert^3\right)$, *space* $O\left(\lvert s\rvert^2\right)$	*Time* $O\left(\lvert s\rvert^3\right)$, *space* $O\left(\lvert s\rvert^2\right)$
#SBP [7]	*Time* $O\left(\lvert s\rvert^{1+\lvert\Sigma\rvert^2+\lvert\Sigma\rvert^3}\right)$, *space* $O\left(\lvert s\rvert^{\lvert\Sigma\rvert^2+\lvert\Sigma\rvert^3}\right)$	**NP** *hard*
#BPS	**NP** *hard for* $\lvert\Sigma\rvert = 2$, *PTAS [7]* 1/3-*approximation in time* $O\left(\lvert s\rvert\right)$ *[6]*	**NP** *hard [7],* 1/3-*approximation in time and space* $O\left(\lvert s\rvert^2\right)$ *[6]*

Theorem 2 *If structures are restricted to be planar, i. e. the graph with the bases of the sequence as nodes and base pairs and backbone links of consecutive bases as edges is required to be planar, pseudoknot prediction under the #BPS scoring scheme is* **NP** *hard for an alphabet of size 4. Conversely, a 1/2-approximation can be found in time* $O\left(|s|^3\right)$



RNA Secondary Structure Prediction Including Pseudoknots, Figure 1
Secondary structure of the *Escherichia coli* α operon mRNA from position 16 to position 127, cf. [5], Figure 1. The backbone of the RNA molecule is drawn as *straight lines* while base pairings are shown with *zigzagged lines*

and space $O\left(|s|^2\right)$ by observing that an optimal pseudoknot free structure is a 1/2-approximation [6].

There are no steric reasons that RNA secondary structures should be planar, and the structure in Fig. 1 is actually non-planar. Nevertheless, known real structures have relatively simple overlapping base pair patterns with very few non-planar structures known. Hence, planarity has been used as a defining restriction on pseudoknotted structures [2,15]. Similar reasoning has lead to development of several algorithms for finding an optimal structure from restricted classes of structures. These algorithms tend to use more realistic scoring schemes, e. g. extensions of the Gibbs free energy model, than the three simple scoring schemes considered above.

Theorem 3 *Pseudoknot prediction for a restricted class of structures including Fig. 2a through Fig. 2e, but not Fig. 2f, can be done in time $O\left(|s|^6\right)$ and space $O\left(|s|^4\right)$* [11].

Theorem 4 *Pseudoknot prediction for a restricted class of planar structures including Fig. 2a through Fig. 2c, but not Fig. 2d through Fig. 2f, can be done in time $O\left(|s|^5\right)$ and space $O\left(|s|^4\right)$* [14].

Theorem 5 *Pseudoknot prediction for a restricted class of planar structures including Fig. 2a and Fig. 2b, but not Fig. 2c through Fig. 2f, can be done in time $O\left(|s|^5\right)$ and space $O\left(|s|^4\right)$ or $O\left(|s|^3\right)$* [1,4] *(methods differ in generality of scoring schemes that can be used).*

Theorem 6 *Pseudoknot prediction for a restricted class of planar structures including Fig. 2a, but not Fig. 2b through Fig. 2f, can be done in time $O\left(|s|^4\right)$ and space $O\left(|s|^2\right)$* [1,8].

Theorem 7 *Recognition of structures belonging to the restricted classes of Theorems 3, 5, and 6, and enumeration of all irreducible cycles (i. e. loops) in such structures can be done in time $O(|s|)$* [3,9].

Applications

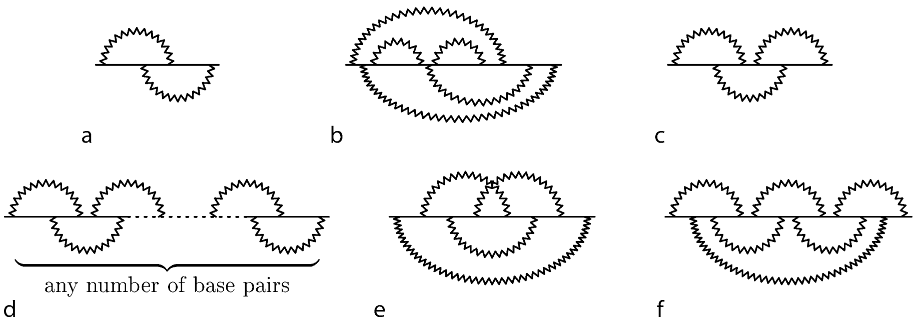
As for the prediction of RNA secondary structures without pseudoknots, the key application of these algorithms are for predicting the secondary structure of individual RNA molecules. Due to the steep complexities of the algorithms of Theorems 3 through 6, these are less well suited for genome scans than prediction without pseudoknots.

Enumerating all loops of a structure in linear time also allows scoring a structure in linear time, as long as the scoring scheme allows the score of a loop to be computed in time proportional to its size. This has practical applications in heuristic searches for good structures containing pseudoknots.

Open Problems

Efficient algorithms for prediction based on restricted classes of structures with pseudoknots that still contain a significant fraction of all known structures is an active area of research. Even using the more theoretical simple $\#SBP$ scoring scheme, developing e. g. an $O\left(|s|^{|\Sigma|}\right)$ algorithm for this problem would be of practical significance. From a theoretical point of view, the complexity of planar structures is the least well understood, with results for only the $\#BPS$ scoring scheme.

Classification of and realistic energy models for RNA secondary structures with pseudoknots are much less developed than for RNA secondary structures without pseudoknots. Several recent papers have been addressing this gap [3,9,12].



RNA Secondary Structure Prediction Including Pseudoknots, Figure 2
RNA secondary structures illustrating restrictions of pseudoknot prediction algorithms. Backbone is drawn as a *straight line* while base pairings are shown with *zigzagged arcs*

Data Sets

PseudoBase at http://biology.leidenuniv.nl/~batenburg/PKB.html is a repository of representatives of most known RNA structures with pseudoknots.

URL to Code

The method of Theorem 3 is available at http://selab.janelia.org/software.html#pknots, of one of the methods of Theorem 5 at http://www.nupack.org, and an implementation applying a slight heuristic reduction of the class of structures considered by the method of Theorem 6 is available at http://bibiserv.techfak.uni-bielefeld.de/pknotsrg/ [10].

Cross References

► RNA Secondary Structure Prediction by Minimum Free Energy

Recommended Reading

1. Akutsu, T.: Dynamic programming algorithms for RNA secondary structure prediction with pseudoknots. Discret. Appl. Math. **104**, 45–62 (2000)
2. Brown, M., Wilson, C.: RNA pseudoknot modeling using intersections of stochastic context free grammars with applications to database search. In: Hunter, L., Klein, T. (eds.) Proceedings of the 1st Pacific Symposium on Biocomputing, 1996, pp. 109–125
3. Condon, A., Davy, B., Rastegari, B., Tarrant, F., Zhao, S.: Classifying RNA pseudoknotted structures. Theor. Comput. Sci. **320**, 35–50 (2004)
4. Dirks, R.M., Pierce, N.A.: A partition function algorithm for nucleic acid secondary structure including pseudoknots. J. Comput. Chem. **24**, 1664–1677 (2003)
5. Gluick, T.C., Draper, D.E.: Thermodynamics of folding a pseudoknotted mRNA fragment. J. Mol. Biol. **241**, 246–262 (1994)
6. Ieong, S., Kao, M.-Y., Lam, T.-W., Sung, W.-K., Yiu, S.-M.: Predicting RNA secondary structures with arbitrary pseudoknots by maximizing the number of stacking pairs. In: Proceedings of the 2nd Symposium on Bioinformatics and Bioengineering, 2001, pp. 183–190
7. Lyngsø, R.B.: Complexity of pseudoknot prediction in simple models. In: Proceedings of the 31th International Colloquium on Automata, Languages and Programming (ICALP), 2004, pp. 919–931
8. Lyngsø, R.B., Pedersen, C.N.S.: RNA pseudoknot prediction in energy based models. J. Comput. Biol. **7**, 409–428 (2000)
9. Rastegari, B., Condon, A.: Parsing nucleic acid pseudoknotted secondary structure: algorithm and applications. J. Comput. Biol. **14**(1), 16–32 (2007)
10. Reeder, J., Giegerich, R.: Design, implementation and evaluation of a practical pseudoknot folding algorithm based on thermodynamics. BMC Bioinform. **5**, 104 (2004)
11. Rivas, E., Eddy, S.: A dynamic programming algorithm for RNA structure prediction including pseudoknots. J. Mol. Biol. **285**, 2053–2068 (1999)
12. Rødland, E.A.: Pseudoknots in RNA secondary structure: Representation, enumeration, and prevalence. J. Comput. Biol. **13**, 1197–1213 (2006)
13. Tabaska, J.E., Cary, R.B., Gabow, H.N., Stormo, G.D.: An RNA folding method capable of identifying pseudoknots and base triples. Bioinform. **14**, 691–699 (1998)
14. Uemura, Y., Hasegawa, A., Kobayashi, S., Yokomori, T.: Tree adjoining grammars for RNA structure prediction. Theor. Comput. Sci. **210**, 277–303 (1999)
15. Witwer, C., Hofacker, I.L., Stadler, P.F.: Prediction of consensus RNA secondary structures including pseudoknots. IEEE Trans. Comput. Biol. Bioinform. **1**, 66–77 (2004)

RNA Secondary Structure Prediction by Minimum Free Energy

2006; Ogurtsov, Shabalina, Kondrashov, Roytberg

Rune B. Lyngsø
Department of Statistics, Oxford University, Oxford, UK

Keywords and Synonyms

RNA Folding

Problem Definition

This problem is concerned with predicting the set of base pairs formed in the native structure of an RNA molecule. The main motivation stems from structure being crucial for function and the growing appreciation of the importance of RNA molecules in biological processes. Base pairing is the single most important factor determining structure formation. Knowledge of the secondary structure alone also provides information about stretches of unpaired bases that are likely candidates for active sites. Early work [7] focused on finding structures maximizing the number of base pairs. With the work of Zuker and Stiegler [17] focus shifted to energy minimization in a model approximating the Gibbs free energy of structures.

Notation

Let $s \in \{A, C, G, U\}^*$ denote the sequence of bases of an RNA molecule. Use $X \cdot Y$ where $X, Y \in \{A, C, G, U\}$ to denote a base pair between bases of type X and Y, and $i \cdot j$ where $1 \le i < j \le |s|$ to denote a base pair between bases $s[i]$ and $s[j]$.

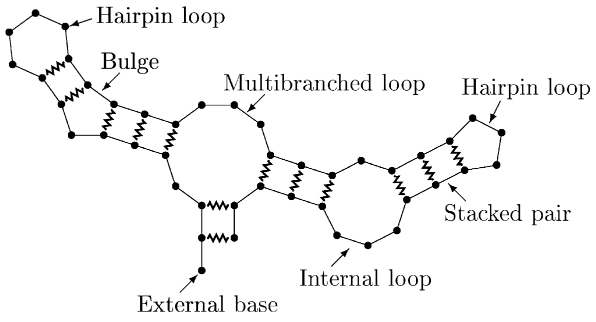
Definition 1 (RNA Secondary Structure) A secondary structure for an RNA sequence s is a set of base pairs $S = \{i \cdot j \mid 1 \le i < j \le |s| \wedge i < j - 3\}$. For $i \cdot j, i' \cdot j' \in S$ with $i \cdot j \neq i' \cdot j'$

- $\{i, j\} \cap \{i', j'\} = \emptyset$ (each base pairs with at most one other base)
- $\{s[i], s[j]\} \in \{\{A, U\}, \{C, G\}, \{G, U\}\}$ (only Watson-Crick and G, U wobble base pairs)
- $i < i' < j \Rightarrow j' < j$ (base pairs are either nested or juxtaposed but not overlapping).

The second requirement, that only canonical base pairs are allowed, is standard but not consequential in solutions to the problem. The third requirement states that the structure does not contain pseudoknots. This restriction is crucial for the results listed in this entry.

Energy Model

The model of Gibbs free energy applied, usually referred to as the nearest-neighbor model, was originally proposed by Tinoco et al. [10,11]. It approximates the free energy by postulating that the energy of the full three dimensional structure only depends on the secondary structure, and



RNA Secondary Structure Prediction by Minimum Free Energy, Figure 1
A hypothetical RNA structure illustrating the different loop types. Bases are represented by *circles*, the RNA backbone by *straight lines*, and base pairs by *zigzagged lines*

that this in turn can be broken into a sum of independent contributions from each loop in the secondary structure.

Definition 2 (Loops) For $i \cdot j \in S$, base k is *accessible* from $i \cdot j$ iff $i < k < j$ and $\neg\exists i' \cdot j' \in S : i < i' < k < j' < j$. The *loop closed by* $i \cdot j$, $\ell_{i \cdot j}$, consists of $i \cdot j$ and all the bases accessible from $i \cdot j$. If $i' \cdot j' \in S$ and i' and j' are accessible from $i \cdot j$, then $i' \cdot j'$ is an interior base pair in the loop closed by $i \cdot j$.

Loops are classified by the number of interior base pairs they contain:

- hairpin loops have no interior base pairs
- stacked pairs, bulges, and internal loops have one interior base pair that is separated from the closing base pair on neither side, on one side, or on both sides, respectively
- multibranched loops have two or more interior base pairs.

Bases not accessible from any base pair are called external. This is illustrated in Fig. 1. The free energy of structure S is

$$\Delta G(S) = \sum_{i \cdot j \in S} \Delta G(\ell_{i \cdot j}) , \tag{1}$$

where $\Delta G(\ell_{i \cdot j})$ is the free energy contribution from the loop closed by $i \cdot j$.

Problem 1 (Minimum Free Energy Structure)
INPUT: *RNA sequence s and specification of ΔG for all loops.*
OUTPUT: $\arg\min_S \{\Delta G(S) \mid S$ *secondary structure for s*$\}$.

Key Results

Solutions are based on using dynamic programming to solve the general recursion

$$V[i,j] = \min_{k\geq 0; i<i_1<j_1<\ldots<i_k<j_k<j} \left\{ \Delta G(\ell_{i\cdot j;i_1\cdot j_1,\ldots,i_k\cdot j_k}) + \sum_{l=1}^{k} V[i_l, j_l] \right\}$$

$$W[i] = \min\left\{ W[i-1], \min_{0<k<i} \{W[k-1] + V[k,i]\} \right\} ,$$

where $´G(\ell_{i\cdot j;i_1\cdot j_1,\ldots,i_k\cdot j_k})$ is the free energy of the loop closed by $i \cdot j$ and interior base pairs $i_1 \cdot j_1, \ldots, i_k \cdot j_k$ and with initial condition $W[0] = 0$. In the following it is assumed that all loop energies can be computed in time $O(1)$.

Theorem 1 *If the free energy of multibranched loops is a sum of*

- *an affine function of the number of interior base pairs and unpaired bases*
- *contributions for each base pair from stacking with either neighboring unpaired bases in the loop or with a neighboring base pair in the loop, whichever is more favorable,*

a minimum free energy structure can be computed in time $O(|s|^4)$ *and space* $O(|s|^2)$ [17].

With these assumptions the time required to handle the multibranched loop parts of the recursion reduces to $O(|s|^3)$. Hence handling the $O(|s|^4)$ possible internal loops becomes the bottleneck.

Theorem 2 *If furthermore the free energy of internal loops is a sum of*

- *a function of the total size of the loop, i. e. the number of unpaired bases in the loop,*
- *a function of the asymmetry of the loop, i. e. the difference in number of unpaired bases on the two sides of the loop,*
- *contributions from the closing and interior base pairs stacking with the neighboring unpaired bases in the loop,*

a minimum free energy structure can be computed in time $O(|s|^3)$ *and space* $O(|s|^2)$ [5].

Under these assumptions the time required to handle internal loops reduces to $O(|s|^3)$. With further assumptions on the free energy contributions of internal loops this can be reduced even further, again making the handling of multibranched loops the bottleneck of the computation.

Theorem 3 *If furthermore the size dependency is concave and the asymmetry dependency is constant for all but O(1) values, a multibranched loop free minimum free energy structure can be computed in time* $O(|s|^2 \log^2 |s|)$ *and space* $O(|s|^2)$ [8].

The above assumptions are all based on the nature of current loop energies [6]. These energies have to a large part been developed without consideration of computational expediency and parameters determined experimentally, although understanding of the precise behavior of larger loops is limited. For multibranched loops some theoretical considerations [4] would suggest that a logarithmic dependency would be more appropriate.

Theorem 4 *If the restriction on the dependency on number of interior base pairs and unpaired bases in Theorem 1 is weakened to any function that depends only on the number of interior base pairs, the number of unpaired bases, or the total number of bases in the loop, a minimum free energy structure can be computed in time $O(n^4)$ and space $O(n^3)$* [13].

Theorem 5 *All the above theorems can be modified to compute a data structure that for any* $1 \leq i < j \leq |s|$ *allows us to compute the minimum free energy of any structure containing $i \cdot j$ in time O(1)* [15].

Applications

Naturally the key application of these algorithms are for predicting the secondary structure of RNA molecules. This holds in particular for sequences with no homologues with common structure, e. g. functional analysis based on mutational effects and to some extent analysis of RNA aptamers. With access to structurally conserved homologues prediction accuracy is significantly improved by incorporating comparative information [2].

Incorporating comparative information seems to be crucial when using secondary structure prediction as the basis of RNA gene finding. As it turns out, the minimum free energy of known RNA genes is not sufficiently different from the minimum free energy of comparable random sequences to reliably separate the two [9,14]. However, minimum free energy calculations is at the core of one successful comparative RNA gene finder [12].

Open Problems

Most current research is focused on refinement of the energy parametrization. The limiting factor of sequence lengths for which secondary structure prediction by the methods described here is still feasible is adequacy of the nearest neighbor approximation rather than computation time and space. Still improvements on time and space complexities are useful as biosequence analyzes are invariably used in genome scans. In particular improvements on Theorem 4, possibly for dependencies restricted to be logarithmic or concave, would allow for more advanced scoring of multibranched loops. A more esoteric open problem is to establish the complexity of computing the minimum free energy under the general formulation of (1), with no restrictions on loop energies except that they are computable in time polynomial in $|s|$.

Experimental Results

With the release of the most recent energy parameters [6] secondary structure prediction by finding a minimum free energy structure was found to recover approximately 73% of the base pairs in a benchmark data set of RNA sequences with known secondary structure. Another independent assessment [1] put the recovery percentage somewhat lower at around 56%. This discrepancy is discussed and explained in [1].

Data Sets

Families of homologous RNA sequences aligned and annotated with secondary structure are available from the Rfam data base at www.sanger.ac.uk/Software/Rfam/. Three dimensional structures are available from the Nucleic Acid Database at ndbserver.rutgers.edu/. An extensive list of this and other data bases is available at www.imb-jena.de/RNA.html.

URL to Code

Software for RNA folding and a range of related problems is available from www.bioinfo.rpi.edu/applications/hybrid/download.php and www.tbi.univie.ac.at/~ivo/RNA/. Software implementing the efficient handling of internal loops of [8] is available from ftp.ncbi.nlm.nih.gov/pub/ogurtsov/Afold.

Cross References

► RNA Secondary Structure Boltzmann Distribution
► RNA Secondary Structure Prediction Including Pseudoknots

Recommended Reading

1. Dowell, R., Eddy, S.R.: Evaluation of several lightweight stochastic context-free grammars for RNA secondary structure prediction. BMC Bioinformatics **5**, 71 (2004)
2. Gardner, P.P., Giegerich, R.: A comprehensive comparison of comparative RNA structure prediction approaches. BMC Bioinformatics **30**, 140 (2004)
3. Hofacker, I.L., Stadler, P.F.: Memory efficient folding algorithms for circular RNA secondary structures. Bioinformatics **22**, 1172–1176 (2006)
4. Jacobson, H., Stockmayer, W.H.: Intramolecular reaction in polycondensations. I. the theory of linear systems. J. Chem. Phys. **18**, 1600–1606 (1950)
5. Lyngsø, R.B., Zuker, M., Pedersen, C.N.S., Fast evaluation of internal loops in RNA secondary structure prediction. Bioinformatics **15**, 440–445 (1999)
6. Mathews, D.H., Sabina, J., Zuker, M., Turner, D.H.: Expanded sequence dependence of thermodynamic parameters improves prediction of RNA secondary structure. J. Mol. Biol. **288**, 911–940 (1999)
7. Nussinov, R., Jacobson, A.B.: Fast algorithm for predicting the secondary structure of single-stranded RNA. Proc. Natl. Acad. Sci. USA **77**, 6309–6313 (1980)
8. Ogurtsov, A.Y., Shabalina, S.A., Kondrashov, A.S., Roytberg, M.A.: Analysis of internal loops within the RNA secondary structure in almost quadratic time. Bioinformatics **22**, 1317–1324 (2006)
9. Rivas, E., Eddy, S.R.: Secondary structure alone is generally not statistically significant for the detection of noncoding RNAs. Bioinformatics **16**, 583–605 (2000)
10. Tinoco, I., Borer, P.N., Dengler, B., Levine, M.D., Uhlenbeck, O.C., Crothers, D.M., Gralla, J.: Improved estimation of secondary structure in ribonucleic acids. Nat. New Biol. **246**, 40–41 (1973)
11. Tinoco, I., Uhlenbeck, O.C., Levine, M.D.: Estimation of secondary structure in ribonucleic acids. Nature **230**, 362–367 (1971)
12. Washietl, S., Hofacker, I.L., Stadler, P.F.: Fast and reliable prediction of noncoding RNA. Proc. Natl. Acad. Sci. USA **102**, 2454–59 (2005)
13. Waterman, M.S., Smith, T.F.: Rapid dynamic programming methods for RNA secondary structure. Adv. Appl. Math. **7**, 455–464 (1986)
14. Workman, C., Krogh, A.: No evidence that mRNAs have lower folding free energies than random sequences with the same dinucleotide distribution. Nucleic Acids Res. **27**, 4816–4822 (1999)
15. Zuker, M.: On finding all suboptimal foldings of an RNA molecule. Science **244**, 48–52 (1989)
16. Zuker, M.: Calculating nucleic acid secondary structure. Curr. Opin. Struct. Biol. **10**, 303–310 (2000)
17. Zuker, M., Stiegler, P.: Optimal computer folding of large RNA sequences using thermodynamics and auxiliary information. Nucleic Acids Res. **9**, 133–148 (1981)

Robotics

1997; (Navigation) Blum, Raghavan, Schieber
1998; (Exploration) Deng, Kameda, Papadimitriou
2001; (Localization) Fleischer, Romanik, Schuierer, Trippen

RUDOLF FLEISCHER
Deptartment of Computer Science and Engineering, Fudan University, Shanghai, China

Keywords and Synonyms

Navigation problem – Search problem
Exploration problem – Mapping problem; Gallery tour problem
Localization problem – Kidnapped robot problem

Problem Definition

Definitions

There are three fundamental algorithmic problems in robotics: exploration, navigation, and localization. *Exploration* means to draw a complete map of an unknown environment. *Navigation* (or *search*) means to find a way to a predescribed location among unknown obstacles. *Localization* means to determine the current position on a known map. Normally, the environment is modeled as a simple polygon with or without holes. To distinguish the underlying combinatorial problems from the geometric problems, the environment may also be modeled as a graph.

Normally, a robot has a compass, i. e., it can distinguish between different directions, and it can measure travel distance. A *blind* (or *tactile*) robot can only sense its immediate surroundings (for example, it only notices an obstacle when it bumps into it; this is also sometimes called *ε-radar*), while a robot *with vision* can see objects far in the distance, unless the view is blocked by opaque obstacles. Robots on graphs are usually blind. In polygonal environments, vision may help to judge the size of an obstacle without moving, but a blind robot can circumvent obstacles with a performance loss of only a factor of nine by using the *lost-cow* doubling strategy [2].

Online Algorithms

An algorithm that tries to approximate an optimal solution by making decisions under a given uncertainty is called an *online algorithm* (see the surveys in [9]). Its performance is measured by the *competitive ratio*, which is the approximation ratio of the online algorithm maximized over all possible input scenarios. In the case of exploration, navigation, and localization, the robot should minimize its travel distance. Therefore, the competitive ratio measures the length of the detour compared to the optimal shortest tour.

A *randomized* online algorithm against an *oblivious adversary* uses randomization on a fixed predetermined input (which is unknown to the online algorithm). In this case, the competitive ratio is a random variable, and it is maximized over all possible inputs.

Exploration

Deng et. al [7] introduced the *gallery tour problem*. Given a polygonal room with polygonal obstacles, an *entry point* s and *exit point* t, a robot with vision needs to travel along a path from s to t such that it can see every point of the perimeter of the polygons. If $s = t$, the problem is known as the *watchman's route problem*. In the online version of the problem, the polygon is initially unknown. The problem becomes easier in rectilinear environments with L_1-metric.

Navigation

Blum et. al [5] studied the problem of a blind robot trying to reach a goal t from a start position s (*point-to-point navigation*) in a scene of non-overlapping axis-parallel rectangles of width at least one. In the *wall problem*, t is an infinite vertical line. In the *room problem*, the obstacles are within a square room with entry door s.

Localization

In the localization problem the robot knows a map of the environment, but not its current position, which it determines by moving around and matching the observed local environment with the given map.

Key Results

Exploration

Theorem 1 ([7]) *The shortest exploration tour in L_1-metric in a known simple rectilinear polygon with n vertices can be computed in time $O(n^3)$.*

Theorem 2 ([15]) *There is a 26.5-competitive online algorithm to explore an unknown simple polygon without obstacles.*

Theorem 3 ([7]) *There is an $O(k + 1)$-competitive online algorithm to explore an unknown simple polygon with k polygonal obstacles.*

Theorem 4 ([1]) *No randomized online algorithm can explore an unknown simple rectilinear polygon with k rectilinear obstacles better than $\Omega(\sqrt{k})$-competitively.*

Navigation

Theorem 5 ([19]) *No online algorithm for the wall problem with n rectangles can be better than $\Omega(\sqrt{n})$-competitive.*

Theorem 6 ([5]) *There are $O(\sqrt{n})$-competitive online algorithms for the wall problem, the room problem, and point-to-point navigation in scenes with n axis-parallel rectangles.*

Theorem 7 ([3]) *There is an optimal $\Theta(\log n)$-competitive online algorithms for the room problem with n axis-parallel rectangles.*

Theorem 8 ([4]) *There are $O(\log n)$-competitive randomized online algorithms against oblivious adversary for the wall problem and point-to-point navigation in scenes with n axis-parallel rectangles.*

Theorem 9 ([16]) *No randomized online algorithm against oblivious adversary for point-to-point navigation between n rectangles can be better than $\Omega(\log\log n)$-competitive.*

Theorem 10 ([5]) *There is a lower bound of $n/8$ for the competitiveness of navigating between n non-convex obstacles. A simple memoryless algorithm achieves a competitive ratio of $50.4 \cdot n$.*

Localization

Theorem 11 ([17]) *No algorithm for localization in geometric trees with n nodes can be better than $\Omega(\sqrt{n})$-competitive.*

Theorem 12 ([11]) *There is an $O(\sqrt{n})$-competitive algorithm for localization in geometric trees with n nodes.*

Applications

Exploration

It is NP-hard to find a shortest exploration tour in a known polygonal environment [7]. Unknown scenes with arbitrary obstacles can be efficiently explored by Lumelsky's `Sightseer Strategy`. Most online exploration algorithms can be transformed into an efficient online algorithm to approximate the *search ratio*, a measure related to the competitive ratio of the navigation problem [10].

The problem of exploring a polygonal environment is closely related to the problem of exploring strongly connected digraphs. Here, the competitive ratio is usually given as a function of the *deficiency* of the graph which is the minimum number of edges that must be added to the graph to make it Eulerian. Eulerian graphs can be explored with a simple optimal 2-competitive algorithm, while graphs of deficiency d can be explored with a competitive factor of $O(d^8)$ [14].

Navigation

In applied robotics, it is common to measure the competitive ratio as a function of the aspect ratio of the obstacle rectangles. Lumelsky's `BUG2` algorithm can navigate between convex obstacles, in the worst case moving at most once around every obstacle, which is optimal.

A robot with a compass can sometimes find the goal exponentially faster than a robot without a compass.

If we need to do several trips in an unknown environment, it may help to use partial map information from previous trips. In particular, the i-th trip between the same two points can be searched $\sqrt{\frac{n}{i}}$-competitively.

Localization

There is a simple k-competitive localization algorithm in polygons and graphs, where k is the number of positions on the map matching the observed environment at the wake-up point.

The *visibility polygon* of a point v is that part of a polygon that a robot can see when sitting at v. One can compute in polynomial time all points of a given simple polygon whose visibility polygon matches a given star polygon.

Computing a shortest localization tour in a known polygon is NP-hard. It can be approximated with a factor of $O(\log^3 n)$, but not better than $\Omega(\log n)$ unless $P = NP$ [18].

Open Problems

Exploration

- A polynomial time algorithm for computing the shortest exploration tour in a known simple polygon without obstacles. Such an algorithm is known for the watchman's route problem.
- A simple online exploration algorithm for simple polygons with tight analysis.
- Exploration and navigation with limited memory.

Navigation

- Online searching among convex polygonal obstacles.
- Three-dimensional navigation, in particular among non-convex obstacles (3d-mazes).

Localization

- A simple algorithm for online localization in trees.
- Online localization in general graphs.
- A randomized online localization algorithm beating deterministic algorithms.

Experimental Results

Exploration

Fleischer and Trippen [13] implemented most known algorithms for exploring a directed graph and demonstrated that the simple (but inferior) greedy algorithms usually outperform the more sophisticated algorithms on random graphs.

Navigation

Coffman and Gilbert [6] implemented eight heuristics for point-to-point navigation in L_1-metric.

Localization

Fleischer and Trippen [12] visualized their localization algorithm in geometric trees.

Cross References

► Alternative Performance Measures in Online Algorithms
► Deterministic Searching on the Line
► Metrical Task Systems
► Randomized Searching on Rays or the Line

Recommended Reading

1. Albers, S., Kursawe, K., Schuierer, S.: Exploring unknown environments with obstacles. Algorithmica **32**(1), 123–143 (2002)
2. Baeza-Yates, R.A., Culberson, J.C., Rawlins, G.J.E.: Searching in the plane. Inf. Comput. **106**(2), 234–252 (1993)
3. Bar-Eli, E., Berman, P., Fiat, A., Yan, P.: Online navigation in a room. J. Algorithms **17**(3), 319–341 (1994)
4. Berman, P., Blum, A., Fiat, A., Karloff, H., Rosén, A., Saks, M.: Randomized robot navigation algorithms. In: Proceedings of the 7th ACM-SIAM Symposium on Discrete Algorithms (SODA'96), 1996, pp. 75–84
5. Blum, A., Raghavan, P., Schieber, B.: Navigating in unfamiliar geometric terrain. SIAM J. Comput. **26**(1), 110–137 (1997)
6. Coffman Jr., E.G., Gilbert, E.N.: Paths through a maze of rectangles. Networks **22**, 349–367 (1992)
7. Deng, X., Kameda, T., Papadimitriou, C.H.: How to learn an unknown environment. J. ACM **45**, 215–245 (1998)
8. Dudek, G., Romanik, K., Whitesides, S.: Localizing a robot with minimum travel. SIAM J. Comput. **27**(2), 583–604 (1998)
9. Fiat, A., Woeginger, G. (eds.) Online Algorithms – The State of the Art. Springer Lecture Notes in Computer Science, vol. 1442. Springer, Heidelberg (1998)
10. Fleischer, R., Kamphans, T., Klein, R., Langetepe, E., Trippen, G.: Competitive online approximation of the optimal search ratio. In: Proceedings of the 12th European Symposium on Algorithms (ESA'04). Lecture Notes in Computer Science, vol. 3221, pp. 335–346. Springer, Heidelberg (2004)
11. Fleischer, R., Romanik, K., Schuierer, S., Trippen, G.: Optimal robot localization in trees. Inf. Comput. **171**, 224–247 (2001)
12. Fleischer, R., Trippen, G.: Optimal robot localization in trees. In: Proceedings of the 16th Annual Symposium on Computational Geometry (SoCG'00), 2000, pp. 373–374. A video shown at the 9th Annual Video Review of Computational Geometry
13. Fleischer, R., Trippen, G.: Experimental studies of graph traversal algorithms. In: Proceedings of the 2nd International Workshop on Experimental and Efficient Algorithms (WEA'03). Lecture Notes in Computer Science, vol. 2647, pp. 120–133. Springer, Heidelberg (2003)
14. Fleischer, R., Trippen, G.: Exploring an unknown graph efficiently. In: Proceedings of the 13th European Symposium on Algorithms (ESA'05). Lecture Notes in Computer Science, vol. 3669, pp. 11–22. Springer, Heidelberg (2005)
15. Hoffmann, F., Icking, C., Klein, R., Kriegel, K.: The polygon exploration problem. SIAM J. Comput. **31**(2), 577–600 (2001)
16. Karloff, H., Rabani, Y., Ravid, Y.: Lower bounds for randomized *k*-server and motion-planning algorithms. SIAM J. Comput. **23**(2), 293–312 (1994)
17. Kleinberg, J.M.: The localization problem for mobile robots. In: Proceedings of the 35th Symposium on Foundations of Computer Science (FOCS'94), 1994, pp. 521–531
18. Koenig, S., Mudgal, A., Tovey, C.: A near-tight approximation lower bound and algorithm for the kidnapped robot problem. In: Proceedings of the 17th ACM-SIAM Symposium on Discrete Algorithms (SODA'06), 2006, pp. 133–142.
19. Papadimitriou, C.H., Yannakakis, M.: Shortest paths without a map. Theor. Comput. Sci. **84**, 127–150 (1991)

Robust Geometric Computation

2004; Li, Yap

CHEE K. YAP, VIKRAM SHARMA
Department of Computer Science, New York University, New York, NY, USA

Keywords and Synonyms

Exact geometric computation Floating-point filter; Dynamic and static filters; Topological consistency

Problem Definition

Algorithms in computational geometry are usually designed under the Real RAM model. In implementing these algorithms, however, fixed-precision arithmetic is used in place of exact arithmetic. This substitution introduces numerical errors in the computations that may lead to nonrobust behavior in the implementation, such as infinite loops or segmentation faults.

There are various approaches in the the literature addressing the problem of nonrobustness in geometric computations; see [9] for a survey. These approaches can be classified along two lines: the **arithmetic approach** and the **geometric approach**.

The arithmetic approach tries to address nonrobustness in geometric algorithms by handling the numerical errors arising because of fixed-precision arithmetic; this can be done, for instance, by using multi-precision arithmetic [6], or by using rational arithmetic whenever possible. In general, all the arithmetic operations, including exact comparison, can be performed on algebraic quantities. The drawback of such a general approach is its inefficiency.

The geometric approaches guarantee that certain geometric properties are maintained by the algorithm. For example, if the Voronoi diagram of a planar point set is being computed then it is desirable to ensure that the output is a planar graph as well. Other geometric approaches are finite resolution geometry [7], approximate predicates and fat geometry [8], consistency and topological approaches [4], and topology oriented approach [13]. The common drawback of these approaches is that they are problem or algorithm specific.

In the past decade, a general approach called the **Exact Geometric Computation** (EGC) [15] has become very successful in handling the issue of nonrobustness in geometric computations; strictly speaking, this approach is subsumed in the arithmetic approaches. To understand the EGC approach, it helps to understand the two parts common to all geometric computations: a *combinatorial structure* characterizing the discrete relations between geometric objects, e. g., whether a point is on a hyperplane or not; and a *numerical part* that consists of the numerical representation of the geometric objects, e. g. the coordinates of a point expressed as rational or floating-point numbers. Geometric algorithms characterize the combinatorial structure by numerically computing the discrete relations (that are embodied in geometric predicates) between geometric objects. Nonrobustness arises when numerical errors in the computations yield an incorrect characterization. The EGC approach ensures that all the geometric predicates are evaluated correctly thereby ensuring the correctness of the computed combinatorial structure and hence the robustness of the algorithm.

Notation

An **expression** E refers to a syntactic object constructed from a given set of operators over the reals $\mathbb{R}$. For example, the set of expressions on the set of operators $\{\mathbb{Z}, +, -, \times, \sqrt{}\}$ is the set of division-free radical expressions on the integers; more concretely, expressions can be viewed as directed acyclic graphs (DAG) where the internal nodes are operators with arity at least one, and the leaves are constants, i. e., operators with arity zero. The value of an expression is naturally defined using induction; note that the value may be undefined. Let E represent both the value of the expression and the expression itself.

Key Results

Following are the key results that have led to the feasibility and success of the EGC approach.

Constructive Zero Bounds

The possibility of EGC approach hinges on the computability of the sign of an expression. For determining the sign of algebraic expressions EGC libraries currently use a numerical approach based upon zero bounds. A **zero bound** $b > 0$ for an expression E is such that absolute value $|E|$ of E is greater than b if the value of E is valid and nonzero. To determine the sign of the expression E, compute an approximation $\tilde{E}$ to E such that $|\tilde{E} - E| < \frac{b}{2}$ if E is valid, otherwise $\tilde{E}$ is also invalid. Then sign of E is the same as the sign of $\tilde{E}$ if $|\tilde{E}| \geq \frac{b}{2}$, otherwise it is zero. A **constructive zero bound** is an effectively computable function B from the set of expressions to real numbers $\mathbb{R}$ such that $B(E)$ is a zero bound for any expression E. For examples of constructive zero bounds, see [2,11].

Approximate Expression Evaluation

Another crucial feature in developing the EGC approach is developing algorithms for approximate expression evaluation, i. e., given an expression E and a relative or absolute precision p, compute an approximation to the value of the expression within precision p. The main computational paradigm for such algorithms is the **precision-driven approach** [15]. Intuitively, this is a downward-upward process on the input expression DAG; propagate precision values down to the leaves in the downward direction; at the leaves of the DAG, assume the ability to approximate the value associated with the leaf to any desired precision; finally, propagate the approximations in the upward direction towards the root. Ouchi [10] has given detailed algorithms for the propagation of "composite precision", a generalization of relative and absolute precision.

Numerical Filters

Implementing approximate expression evaluation requires multi-precision arithmetic. But efficiency can be gained by exploiting machine floating-point arithmetic, which is fast and optimized on current hardware. The basic idea is to to check the output of machine evaluation of predicates, and fallback on multi-precision methods if the check fails. These checks are called numerical filters; they certify certain properties of computed numerical values, such as their sign. There are two main classifications of numerical filters: *static filters* are those that can be mostly computed at compile time, but they yield overly pessimistic error bounds and thus are less effective; *dynamic filters* are implemented during run time and even though they have higher costs they are much more effective than static filters, i. e., have better estimate on error bounds. See Fortune and van Wyk [5].

Applications

The EGC approach has led to the development of libraries, such as LEDA Real and CORE, that provide EGC number types, i. e., a class of expressions whose signs are guaranteed. CGAL, another major EGC Library that provides robust implementation of algorithms in computational geometry, offers various specialized EGC number types, but for general algebraic numbers it can also use LEDA Real or CORE.

Open Problems

1. An important challenge from the perspective of efficiency for EGC approach is high degree algebraic computation, such as those found in Computer Aided Design. These issues are beginning to be addressed, for instance [1].
2. The *fundamental problem of EGC* is the **zero problem**: given any set of real algebraic operators, decide whether any expression over this set is zero or not. The main focus here is on the decidability of the zero problem for non-algebraic expressions. The importance of this problem has been highlighted by Richardson [12]; recently some progress has been made for special non-algebraic problems [3].
3. When algorithms in EGC approach are embedded in larger application systems (such as mesh generation systems), the output of one algorithm needs to be cascaded as input to another; the output of such algorithms may be in high precision, so it is desirable to reduce the precision in the cascade. The geometric version of this problem is called the **geometric rounding problem**: given a consistent geometric object in high precision, "round" it to a consistent geometric object at a lower precision.
4. Recently a computational model for the EGC approach has been proposed [14]. The corresponding complexity model needs to be developed. Standard complexity analysis based on input size is inadequate for evaluating the complexity of real computation; the complexity should be expressed in terms of the output precision.

URL to Code

1 `Core Library`: http://www.cs.nyu.edu/exact
2 `LEDA`: http://www.mpi-sb.mpg.de/LEDA
3 `CGAL`: http://www.cgal.org

Recommended Reading

1. Berberich, E., Eigenwillig, A., Hemmer, M., Hert, S., Schmer, K. M., Schmer, E.: A computational basis for conic arcs and boolean operations on conic polygons. In: 10th European Symposium on Algorithms (ESA'02), pp. 174–186, (2002) Lecture Notes in CS, No. 2461
2. Burnikel, C., Funke, S., Mehlhorn, K., Schirra, S., Schmitt, S.: A separation bound for real algebraic expressions. In: Lecture Notes in Computer Science, pp. 254–265. Springer, vol 2161 (2001)
3. Chang, E.C., Choi, S.W., Kwon, D., Park, H., Yap, C.: Shortest Paths for Disc Obstacles is Computable. In: Gao, X.S., Michelucci, D. (eds.) Special Issue on Geometric Constraints. Int. J. Comput. Geom. Appl. **16**(5–6), 567–590 (2006), Also appeared in Proc. 21st ACM Symp. Comp. Geom., pp. 116–125 (2005)
4. Fortune, S.J.: Stable maintenance of point-set triangulations in two dimensions. IEEE Found. Comput. Sci.: **30**, 494–499 (1989)
5. Fortune, S.J., van Wyk, C.J.: Efficient exact arithmetic for computational geometry. In: Proceeding 9th ACM Symposium on Computational Geometry, pp. 163–172 (1993)
6. Gowland, P., Lester, D.: Asurvey of exact arithmetic implementations. In: Blank, J., Brattka, V., Hertling, P. (eds.) Computability and Complexity in Analysis, pp. 30–47. Springer, 4th International Workshop, CCA 2000, Swansea, UK, September 17–19, (2000), Selected Papers. Lecture Notes in Computer Science, No. 2064
7. Greene, D.H., Yao, F.F.: Finite-resolution computational geometry. IEEE Found. Comput. Sci. **27**, 143–152 (1986)
8. Guibas, L., Salesin, D., Stolfi, J.: Epsilon geometry: building robust algorithms from imprecise computations. ACM Symp Comput. Geometr. **5**, 208–217 (1989)
9. Li, C., Pion, S., Yap, C.K.: Recent progress in Exact Geometric Computation. J. Log. Algebr. Program. **64**(1), 85–111 (2004)
10. Ouchi, K.: Real/Expr: Implementation of an exact computation package. Master's thesis, New York University, Department of Computer Science, Courant Institute, January (1997). URL http://cs.nyu.edu/exact/doc/
11. Pion, S., Yap, C.: Constructive root bound method for k-ary rational input numbers, September, (2002). Extended Abstract. Submitted, (2003) ACM Symposium on Computational Geometry
12. Richardson, D.: How to recognize zero. J. Symb. Comput. **24**, 627–645 (1997)
13. Sugihara, K., Iri, M., Inagaki, H., Imai, T.: Topology-oriented implementation—an approach to robust geometric algorithms. Algorithmica **27**, 5–20 (2000)
14. Yap, C.K.: Theory of Real Computation according to EGC. To appear in LNCS Volume based on talks at a Dagstuhl Seminar "Reliable Implementation of Real Number Algorithms: Theory and Practice", Jan 8–13, (2006)
15. Yap, C.K., Dubé, T.: The exact computation paradigm. In: Du, D.Z., Hwang, F.K.: (eds.) Computing in Euclidean Geometry, 2nd edn., pp. 452–492. World Scientific Press, Singapore (1995)

Robustness

- ▶ Connectivity and Fault-Tolerance in Random Regular Graphs
- ▶ Distance-Based Phylogeny Reconstruction (Optimal Radius)
- ▶ False-Name-Proof Auction

Routing

2003; Azar, Cohen, Fiat, Kaplan, Räcke

JÓZSEF BÉKÉSI, GÁBOR GALAMBOS
Department of Computer Science, Juhász Gyula Teachers Training College, Szeged, Hungary

Keywords and Synonyms

Routing algorithms; Network flows; Oblivious routing

Problem Definition

One of the most often used techniques in modern computer networks is routing. *Routing* means selecting paths in a network along which to send data. Demands usually randomly appear on the nodes of a network, and routing algorithms should be able to send data to their destination. The transfer is done through intermediate nodes, using the connecting links, based on the topology of the network. The user waits for the network to guarantee that it has the required capacity during data transfer, meaning that the network behaves like its nodes would be connected directly by a physical line. Such service is usually called the *permanent virtual circuit (PVC)* service. To model real life situations, assume that demands arrive *on line*, given by source and destination points, and capacity (bandwidth) requirements.

Similar routing problems may occur in other environments, for example in parallel computation. In this case there are several processors connected together by wires. During an operation some data appear at given processors which should be sent to specific destinations. Thus, this also defines a routing problem. However, this paper mainly considers the network model, not the parallel computer one.

For any given situation there are several routing possibilities. A natural question is to ask which is the best possible algorithm. To find the best algorithm one must define an objective function, which expresses the effectiveness of the algorithm. For example, the aim may be to reduce the load of the network. Load can be measured in different ways, but to measure the utilization percent of the nodes or the links of the network is the most natural. In the online setting, it is interesting to compare the behavior of a routing algorithm designed for a specific instance to the best possible routing.

There are two fundamental approaches towards routing algorithms. The first approach is to route *adaptively*, i. e. depending on the actual loads of the nodes or the links. The second approach is to route *obviously*, without using any information about the current state of the network. Here the authors survey only results on oblivious routing algorithms.

Notations and Definitions

A mathematical model of the network routing problem is now presented.

Let $G(V, E, c)$ be a capacitated network, where V is the set of nodes and E is the set of edges with a capacity function $c : E \to R^+$. Let $|V| = n, |E| = m$. It can be assumed that G is directed, because if G is undirected then for each undirected edge $e = (u, v)$ two new nodes x, y and four new directed edges $e_1 = (u, x), e_2 = (v, x), e_3 = (y, u)$, $e_4 = (y, u)$ with infinite capacity may be added to the graph. If e is considered as an undirected edge with the same capacity then a directed network equivalent to the original one is received.

Definition 1 A set of functions $f := \{f_{ij} | i, j \in V, f_{ij} : E(G) \to R^+\}$ is called a **multi–commodity flow** if

$$\sum_{e \in E_k^+} f_{ij}(e) = \sum_{e \in E_k^-} f_{ij}(e)$$

holds for all $k \neq i, k \neq j$, where $k \in V$ and E_k^+, E_k^- are the set of edges coming out from k and coming into k resp. Each function f_{ij} defines a **single–commodity flow** from i to j.

Definition 2 The **value** of a multi–commodity *flow* is an $n \times n$ matrix $T_f = (t_{ij}^f)$, where

$$t_{ij}^f = \sum_{e \in E_i^+} f_{ij}(e) - \sum_{e \in E_i^-} f_{ij}(e) ,$$
$$\text{if } i \neq j \text{ and } v_{ii}^f = 0 , \text{ for all } i, j \in V .$$

Definition 3 Let D be a nonnegative $n \times n$ matrix where the diagonal entries are 0. D is called as **demand matrix**. The **flow** on an edge $e \in E$ routing the demand matrix D by routing r is defined by

$$\text{flow}(e, r, D) = \sum_{i,j \in V} d_{ij} r_{ij}(e) ,$$

while the **edge congestion** is

$$\mathrm{con}(e, r, D) = \frac{\mathrm{flow}(e, r, D)}{c(e)} .$$

The **congestion** of demand D using routing r is

$$\mathrm{con}(r, D) = \max_{e \in E} \mathrm{con}(e, r, D) .$$

Definition 4 A multi–commodity flow r is called **routing** if $t^r_{ij} = 1$, and if $i \neq j$ for all $i, j \in V$.

Routing represents a way of sending information over a network. The real load of the edges can be represented by scaling the edge congestions with the demands.

Definition 5 The **oblivious performance ratio** P_r of routing r is

$$P_r = \sup_D \left\{ \frac{\mathrm{con}(r, D)}{\mathrm{opt}(D)} \right\}$$

where $opt(D)$ is the optimal congestion which can be achieved on D. The **optimal oblivious routing ratio** for a network G is denoted by $opt(G)$, where

$$\mathrm{opt}(G) = \min_r P_r$$

Problem
INPUT: *A capacitated network $G(V, E, c)$.*
OUTPUT: *An oblivious routing r, where P_r is minimal.*

Key Results

Theorem 1 *There is a polynomial time algorithm that for any input network G (directed or undirected) finds the optimal oblivious routing ratio and the corresponding routing r.*

Theorem 2 *There is a directed graph G of n vertices such that opt(G) is at least $\Omega(\sqrt{n})$.*

Applications

Most importantly, with these results one can efficiently calculate the best routing strategy for a network topology with capacity constraints. This is a good tool for network planning. The effectiveness of a given topology can be tested without any knowledge of the the network traffic using this analysis.

Many researchers have investigated the variants of routing problems. For surveys on the most important models and results, see [10] and [11]. Oblivious routing algorithms were first analyzed by Valiant and Brebner ([15]). Here, they considered the parallel computer model and investigated specific architectures, like hypercube, square grids, etc. Borodin and Hopcroft investigated general networks ([6]). They showed that such simple deterministic strategies like oblivious routing can not be very efficient for online routing and proved a lower bound on the competitive ration of oblivious algorithms. This lower bound was later improved by Kaklamanis et al. ([9]), and they also gave an optimal oblivious deterministic algorithm for the hypercube.

In 2002, Räcke constructed a polylog competitive randomized algorithm for general undirected networks. More precisely, he proved that for any demand there is a routing such that the maximum edge congestion is at most polylog(n) times the optimal congestion for this demand ([12]). The work of Azar et al. extends this result by giving a polynomial method for calculating the optimal oblivious routing for a network. They also prove that for directed networks no logarithmic oblivious performance ratio exists. Recently, Hajiaghayi et al. present an oblivious routing algorithm which is $O\left(\log^2 n\right)$-competitive with high probability in directed networks ([8]).

A special online model has been investigated in [5], where the authors define the so called "repeated game" setting, where the algorithm is allowed to chose a new routing in each day. This means that it is oblivious to the demands, that will occur the next day. They present an $1 + \varepsilon$-competitive algorithm for this model.

There are better algorithms for the adaptive case, for example in [2]. For the offline case Raghavan and Thomson gave an efficient algorithm in [13].

Open Problems

The authors investigated edge congestion in this paper, but in practice, node congestion may be interesting as well. Node congestion means the ratio of the total traffic traversing a node to its capacity. Some results can be found for this problem in [7] and in [3]. It is an open problem whether this method used for edge congestion analysis can be applied for such a model. Another interesting open question may be whether there is a more efficient algorithm to compute the optimal oblivious performance ratio of a network ([1,14]).

Experimental Results

The authors applied their method on ISP network topologies and found that the calculated optimal oblivious ratios are surprisingly low, between 1.4 and 2. Other research dealing with this question found similar results ([1,14]).

Cross References

- ▶ Approximate Maximum Flow Construction
- ▶ Direct Routing Algorithms
- ▶ Load Balancing
- ▶ Mobile Agents and Exploration
- ▶ Oblivious Routing
- ▶ Probabilistic Data Forwarding in Wireless Sensor Networks

Recommended Reading

1. Applegate, D., Cohen, E.: Making routing robust to changing traffic demands: algorithms and evaluation. IEEE/ACM Trans Netw **14**(6), 1193–1206 (2006). doi:10.1109/TNET.2006.886296
2. Aspnes, J., Azar, Y., Fiat, A., Plotkin, S., Waarts, O.: On-line routing of virtual circuits with applications to load balancing and machine scheduling. J. ACM **44**(3), 486–504 (1997)
3. Azar, Y., Chaiutin, Y.: Optimal node routing. In: Proceedings of the 23rd International Symposium on Theoretical Aspects of Computer Science, 2006, pp. 596–607
4. Azar, Y., Cohen, E., Fiat, A., Kaplan, H. Räcke, H.: Optimal oblivious routing in polynomial time. In: Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing, 2003, pp. 383–388
5. Bansal, N., Blum, A., Chawla, S.: Meyerson, A.: Online oblivious routing. In: Proceedings of the 15th Annual ACM Symposium on Parallel Algorithms, 2003, pp. 44–49
6. Borodin, A., Hopcroft, J.E.: Routing, merging and sorting on parallel models of computation. J. Comput. Syst. Sci. **30**(1), 130–145 (1985)
7. Hajiaghayi, M.T., Kleinberg, R.D., Leighton, T., Räcke, H.: Oblivious routing on node-capacitated and directed graphs. In: Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms, 2005, pp. 782–790
8. Hajiaghayi, M.T., Kim, J.H., Leighton, T., Räcke, H.: Oblivious routing in directed graphs with random demands. In: Proceedings of the 37th Annual ACM Symposium on Theory of Computing, 2005, pp. 193–201
9. Kaklamanis, C., Krizanc, D., Tsantilas, A.: Tight bounds for oblivious routing in the hypercube. In: Proc. 2nd Annual ACM Symposium on Parallel Algorithms and Architectures, 1990, pp. 31–36
10. Leighton, F.T.: Introduction to Parallel Algorithms and Architectures Arrays, Trees, Hypercubes. Morgan Kaufmann Publishers, San Fransisco (1992)
11. Leonardi, S.: On-line network routing. In: Fiat, A., Woeginger, G. (eds.) Online Algorithms – The State of the Art. Chap. 11, pp. 242–267. Springer, Heidelberg (1998)
12. Räcke, H.: Minimizing Congestions in General Networks. In: Proceedings of the 43rd Symposium on Foundations of Computer Science, 2002, pp. 43–52
13. Raghavan, P., Thompson, C.D.: Randomized rounding: a technique for provably good algorithms and algorithmic proofs. Combinatorica **7**, 365–374 (1987)
14. Spring, N., Mahajan, R., Wetherall, D.: Measuring ISP topologies with Rocketfuel. In: Proceedings of the ACM SIGCOMM'02 Conference. ACM, New York (2002)
15. Valiant, L.G., Brebner, G.: Universal schemes for parallel communication. In: Proceedings of the 13th ACM Symposium on Theory of Computing, 1981, pp. 263–277

Routing in Geometric Networks

2003; Kuhn, Wattenhofer, Zhang, Zollinger

LESZEK GĄSIENIEC, CHANG SU, PRUDENCE WONG
Department of Computer Science, University of Liverpool, Liverpool, UK

Keywords and Synonyms

Geometric routing; Geographic routing; Location-based routing

Problem Definition

Network Model/Communication Protocol

In geometric networks, the nodes are embedded into Euclidean plane. Each node is aware of its geographic location, i. e., it knows its (x, y) coordinates in the plane.

Each node has the same transmission range, i. e., if a node v is within the transmission range of another node u, the node u can transmit to v directly and vice versa. Thus, the network can be modeled as an undirected graph $G = (V, E)$, where two nodes $u, v \in V$ are connected by an edge $(u, v) \in E$ if u and v are within their transmission ranges. Such two nodes are called *neighboring nodes* or simply *neighbors*. If two nodes are outside of their transmission ranges a multi-hop transmission is involved, i. e., the two nodes must communicate via intermediate nodes.

The cost $c(e)$ of sending a message over an edge $e \in E$ to a neighboring node has been modeled in many different ways. The most common ones include: the *hop (link) metric* ($c(e) = 1$), the *Euclidean metric* ($c(e) = |e|$), where $|e|$ is the Euclidean length of the edge e, and the *energy metric* ($c(e) = |e|^\alpha$ for $\alpha \geq 2$).

In geometric networks there is no fixed infrastructure nor a central server. I.e., all the nodes act as hosts as well as routers. The topology of the network is unknown to the nodes apart from their direct neighborhood, i. e., each node is aware of its own location as well as the coordinates of its neighbors. The nodes need to discover and maintain routes (involved in multi-hop transmissions) by themselves in a distributed manner. It is also very often assumed (in the context of sensor networks) that each node has limited memory and power.

Geometric routing is to route a message from a *source node* s to a destination t using geographic location information, i. e., the coordinates of the nodes. It is assumed that the source node knows the coordinates of the destination node. A dedicated external location service is used for the source node to obtain this information [8]. The routing protocol consists of a sequence of communication steps. During each step, both the label of a unique transmitting node as well as the label of one of its neighbors who is expected to receive the transmitted message are specified by the routing protocol. Geometric routing is uniform in the context that all nodes execute the same protocol when deciding to which other node to forward a message.

Three classes of geometric routing algorithms are considered: *on-line geometric routing*, *off-line geometric routing* and *dynamic geometric routing*. In the context of all three classes, the focus is on routing the message from the source node to the destination using as small number of communication steps as possible. Note that the number of communication steps corresponds to the total number of transmissions. Thus by minimizing the number of communication steps, the number of transmissions is also minimized resulting in reduced power consumption. In what follows a list of combinatorial and algorithmic definitions commonly used in the context of geometric routing is given.

Planar Graph A graph $G = (V, E)$ is *planar* if nodes in V can be *embedded* into a 2-dimensional Euclidean Space $\mathcal{R}^2$, i. e., each node in V obtains a unique coordinates and an edge is drawn between every pair of nodes in E, in such way the resulting edges do not cross each other in $\mathcal{R}^2$.

Unit-Disk Graph (UDG) is defined to be a graph $G = (V, E)$ embedded into $\mathcal{R}^2$ where two nodes $u, v \in V$ are connected by an edge e if the Euclidean distance between u and v, denoted by $|u, v|$, is not greater than one.

λ-Precision/$\Omega(1)$ Model or Civilized Graph is defined to be a graph $G = (V, E)$ embedded into $\mathcal{R}^2$ where for any fixed $\lambda{>}0$, two nodes $u, v \in V$ are of a distance at least λ apart.

Gabriel Graph (GG) is defined to be a graph $G = (V, E)$ embedded into $\mathcal{R}^2$ where for any $u, v \in V$ an edge $(u, v) \in E$ if u and v are the only nodes in V belonging to the circle with (u, v) as diameter.

Delaunay Triangulation Δ of a set of nodes V embedded into $\mathcal{R}^2$ is the geometric dual of the *Voronoi diagram* [9] of V, where any two nodes in V are linked by an edge in Δ if their corresponding cells in the Voronoi diagram are incident. A Delaunay triangulation Δ is *unit* if it contains edges of length at most one.

The Right Hand Principle is a rule used by graph traversal algorithms that primarily chooses the first edge to the right while moving towards the destination.

Heap-Like Structure Let $G = (V, E)$ be an undirected planar graph, s.t., each node in V contains some numerical value. A *heap-like structure* is a BFS tree T spanning all nodes in G, s.t., for every node v other than the root, the value stored at v is smaller than the value stored at v's parent.

Systems of clusters [2] Let $G = (V, E)$ be an undirected planar graph with $|V| = n$ and radius R. One can construct a system of families of clusters $F(0), F(1), \ldots, F(\log R)$, s.t., (a) the diameter of each cluster in $F(i)$ is $O(2^i \log n)$, (b) every node belongs to at most $O(\log n)$ clusters, and (c) for any two nodes whose distance in G is $2^{i-1} < d \le 2^i$, there exists at least one cluster in $F(i)$ that contains the two nodes.

Key Result and Applications

The key results on geometric routing rely on the following lemmas about Delaunay triangulation, planar graph and unit disk graph.

Lemma 1 ([9]) *The Delaunay triangulation Δ for a set of points V of cardinality n can be computed locally in time $O(n \log n)$.*

Lemma 2 ([4]) *Consider any $s, t \in V$. Assume x and y are two points such that s, x and y belong to a Delaunay triangulation Δ. And let α and β be the angles formed by segments $\overline{xs}$ and $\overline{st}$, and by segments $\overline{ts}$ and $\overline{sy}$ respectively. If $\alpha < \beta$, then $|xs| < |st|$. Otherwise $|ys| < |st|$.*

Lemma 3 *Let $G = (V, E)$ be a planar graph embedded into $\mathcal{R}^2$ and $s, t \in V$. Further, let x_i be the closest to t intersection point defined by some edge e_i belonging to some face F_i and the line segment $\overline{st}$. Similarly, let x_{i+1} be the closest to t intersection point defined by some edge belonging to face F_{i+1} and the line segment $\overline{st}$, where F_{i+1} is the face incident to F_i via edge e_i. Then $|x_i, t| > |x_{i+1}, t|$.*

Lemma 4 ([6]) *Let $G = (V, E)$ be a planar civilized graph embedded into $\mathcal{R}^2$. Any ellipse with major axis c covers at most $O(c^2)$ nodes and edges.*

Lemma 5 ([5]) *Let R be a convex region in $\mathcal{R}^2$ with area $A(R)$ and perimeter $P(R)$, and let $V \subset R$. If the unit disk graph of V has maximum degree k, the number of nodes in V is bounded by $|V| \le 8(k + 1)(A(R) + P(R) + \pi)/\pi$.*

Lemma 6 ([2]) *The number of transmissions required to construct a heap-like structure and the system of clusters for a planar graph G is bounded by $O(nD)$ and $O(n^2 D)$, respectively, where n is the number of nodes and D is the diameter of G.*

Applications

On-Line Geometric Routing

On-line geometric routing is based on very limited control information possessed by the routed message and the local information available at the network nodes. This results in natural scalability of geometric routing. It is also assumed that the network is static, i. e., the nodes do not move and no edges disappear nor (re)appear.

Compass Routing I (*CR-I*) [4] is a greedy procedure based on Delaunay triangulation and the observation from Lemma 2, where during each step the message is always routed to a neighboring node which is closer to the destination t. Unfortunately, the message may eventually end up in a local minimum (*dead end*) where all neighbors are further away from t. *CR-I* is very simple. Also computation of Delaunay triangulation is local and cheap, see Lemma 1. However, the algorithm does not guarantee successful delivery.

Compass Routing II (*CR-II*) [1,4] is the first geometric routing algorithm based on the right hand principle and the observation from Lemma 3 which guarantee successful delivery in any graph embedded into $\mathcal{R}^2$. The algorithm is also known as *Face Routing* since the routed message traverses along perimeters of faces closer and closer to the destination. In convex graph, the segment $\overline{st}$ intersects the perimeter of any face at most twice. Thus when the routed message hits the first edge e that intersects $\overline{st}$, it immediately changes the face to the other side of e. In consequence, every edge in each face is traversed at most twice. However, in general graph the routed message has to visit all edges incident to the face. This is to find the closest intersection point x_i to the destination t. In this case each edge can be visited even 4 times. However if after the traversal of all edges the routed message chooses the shorter path to x_i (rather than using the right hand principle), the amortized traversal cost of each edge is 3 [1]. The proof of correctness follows from Lemma 3.

Theorem 7 ([1]) *Compass Routing II guarantees successful delivery in planar graphs using $O(n)$ time where n is the number of nodes in the network.*

Adaptive Face Routing (*AFR*) [6] is an asymptotically optimal geometric routing in planar civilized graphs. The algorithm attempts to estimate the length c of the shortest path between s and t by $\widehat{c}$ (starting with $\widehat{c} = 2|\overline{st}|$ and doubling it in every consecutive round). In each round, the face traversal is restricted to the region formed by the ellipse with the major axis $\widehat{c}$ centered in $\overline{st}$. In AFR each edge is traversed at most 4 times, and the time complexity of *AFR* is $O(c^2)$, see Lemma 4. The corresponding lower bound is also provided in [6].

Theorem 8 ([6]) *The time complexity $O(c^2)$, where c is the length of the shortest path between s and t, is asymptotically optimal in civilized Unit Disk Graphs possessing Gabriel Graph properties.*

Geometric Ad-hoc Routing (*GOAFR*$^+$) [5] has provably good theoretical and practical performance. Due to Lemma 5, rather non-practical Ω (1) assumption can be dropped. *GOAFR*$^+$ combines greedy routing and face routing algorithms. The algorithm starts with the greedy routing *CR-I* and when the routed message enters a local minimum (*dead end*), it switches to *Face Routing*.

However, *GOAFR*$^+$ intends to return to greedy routing as early as possible via application of *early fallback* technique. The simulations show that *GOAFR*$^+$ outperforms *GOAFR* and *GOAFR*$_{FC}$ considered in [7] in the average case.

Theorem 9 ([2]) *GOAFR*$^+$ *has the optimal time complexity $O(c^2)$ in any Unit Disk Graphs possessing Gabriel Graph properties.*

Off-Line Geometric Routing

In off-line geometric routing, the routing stage is preceded by the preprocessing stage, when several data structures are constructed on the basis of the input graph G. This is to speed up the routing phase. The preprocessing is worthwhile if it is followed by further frequent queries.

Single-Source Queries [2] is a routing mechanism that allows to route messages from a distinguished source s to any other node t in the network in time $O(c)$, where c is the distance between s and t in G. The routing procedure is based on indirect addressing mechanism implemented in a heap-like structure that can be efficiently computed, see Lemma 6.

Multiple-Source-Queries [2] is an extension of the single-source querying mechanism that provides $O(c \log n)$-time routing between any pair of nodes located at distance c in G, where n is the number of nodes in G. The extension is based on the system of clusters that can be computed efficiently, see Lemma 6.

Theorem ([5]) *After preprocessing, single-source queries take time $O(c)$ and multiple-source queries take time $O(c \log n)$ in Unit Disk Graphs possessing Gabriel Graph properties.*

Dynamic Geometric Routing

Geometric Routing in Graphs with Dynamic Edges [3] applies to the model in which the nodes are fault-free and stationary but the edges alternate their status between *active* and *inactive*. However, it is assumed that despite dynamic changes in the topology the network always remains connected. In this model *Timestamp-Traversal* routing algorithm combines the use of the global time and the starting time of the routing to traverse a spanning subgraph containing only stable links.

An alternative solution called *Tethered-Traversal* is based on the observation that (re)appearing edges potentially shorten the traversal paths, where the time/space complexity of the routing procedure is linear in the number of nodes n.

Open Problems

Very little is known about space efficient on-line routing in static *directed* graphs. Also the current bounds in dynamic geometric routing appear to be far from optimal.

Cross References

► Communication in Ad Hoc Mobile Networks Using Random Walks
► Minimum k-Connected Geometric Networks

Recommended Reading

1. Bose, P., Morin, P., Stojmenovic, I., Urrutia, J.: Routing with guaranteed delivery in ad hoc wireless networks. In: Proceedings of the Third International Workshop on Discrete Algorithm and Methods for Mobility, Seattle, Washington, Aug 1999, pp. 48–55
2. Gasieniec, L., Su, C., Wong, P.W.H., Xin, Q.: Routing via single-source and multiple-source queries in static sensor networks. J. Discret. Algorithm **5**(1), 1–11 (2007). A preliminary version of the paper appeared in IPDPS'2005
3. Guan, X.Y.: Face traversal routing on edge dynamic graphs. In: Proceedings of the Nineteenth International Parallel and Distributed Processing Symposium, Denver, Colorado, April 2005
4. Kranakis, E., Singh, H., Urrutia, J.: Compass routing on geometric networks. In: Proceedings of the Eleventh Canadian Conference on Computational Geometry, Vancover, BC, Canada, Aug 1999, pp. 51–54
5. Kuhn, F., Wattenhofer, R., Zhang, Y., Zollinger, A.: Geometric ad-hoc routing: Of theory and practice. In: Proceedings of the Twenty-Second ACM Symposium on the Principles of Distributed Computing, Boston, Massachusetts, July 2003, pp. 63–72
6. Kuhn, F., Wattenhofer, R., Zollinger, A.: Asymptotically optimal geometric mobile ad-hoc routing. In: Proceedings of the Sixth International Workshop on Discrete Algorithm and Methods for Mobility, Atlanta, Georgia, USA, Sept 2002, pp. 24–33
7. Kuhn, F., Wattenhofer, R., Zollinger, A.: Worst-case optimal and average-case efficient geometric ad-hoc routing. In: Proceedings of the Fourth ACM International Symposium on Mobile Ad Hoc Networking and Computing, Annapolis, Maryland, June 2003, pp. 267–278
8. Li, J., Jannotti, J., De Couto, D.S.J., Karger, D.R., Morris, R.: A scalable location service for geographic ad hoc routing. In Proceedings of the Sixth International Conference on Mobile Computing and Networking, Boston, Massachusetts, Aug 2000, pp. 120–130
9. Li, M., Lu, X.C., Peng, W.: Dynamic delaunay triangulation for wireless ad hoc network. In Proceedings of the Sixth International Workshop on Advanced Parallel Processing Technologies, Hong Kong, China, Oct 2005, pp. 382–389

Routing in Road Networks with Transit Nodes
2007; Bast, Funke, Sanders, Schultes

DOMINIK SCHULTES
Institute for Computer Science,
University of Karlsruhe, Karlsruhe, Germany

Keywords and Synonyms

Shortest paths

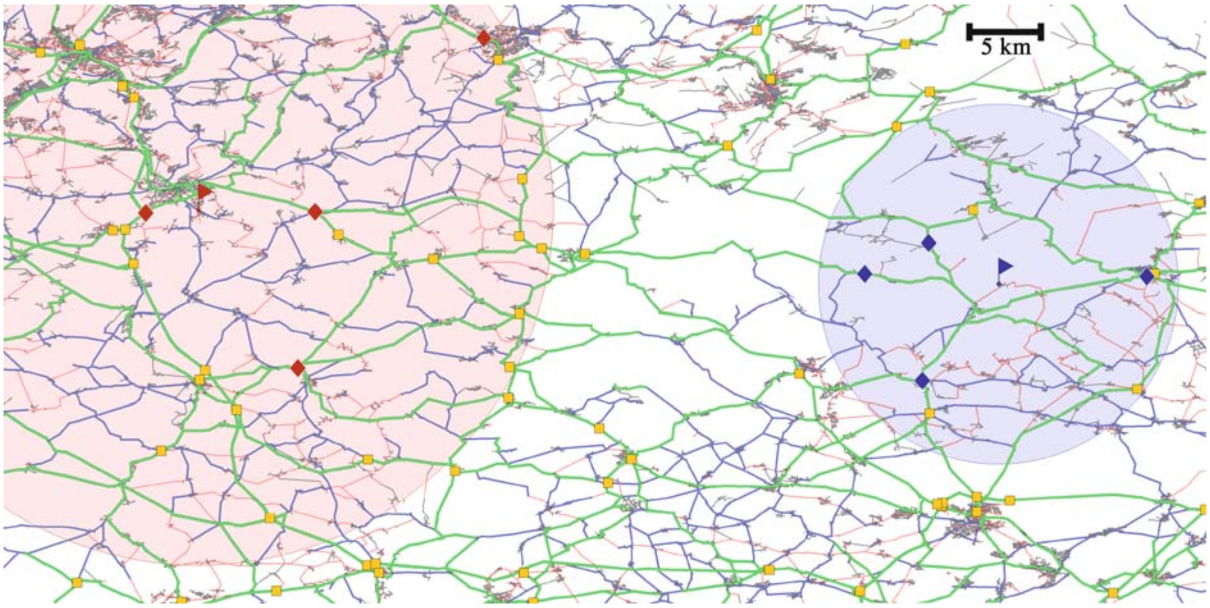
Problem Definition

For a given directed graph $G = (V, E)$ with non-negative edge weights, the problem is to compute a shortest path in G from a source node s to a target node t for given s and t. Under the assumption that G does not change and that a lot of source-target queries have to be answered, it pays to invest some time for a preprocessing step that allows for very fast queries. As output, either a full description of the shortest path or only its length $d(s, t)$ is expected—depending on the application.

Dijkstra's classical algorithm for this problem [4] iteratively visits all nodes in the order of their distance from the source until the target is reached. When dealing with very large graphs, this general algorithm gets too slow for many applications so that more specific techniques are needed that exploit special properties of the particular graph. One practically very relevant case is routing in road networks where junctions are represented by nodes and road segments by edges whose weight is determined by some weighting of, for example, expected travel time, distance, and fuel consumption. Road networks are typically sparse (i. e., $|E| = O(|V|)$), almost planar (i. e., there are only a few overpasses), and hierarchical (i. e., more or less 'important' roads can be distinguished). An overview on various speedup techniques for this specific problem is given in [7].

Key Results

Transit-node routing [2,3] is based on a simple observation intuitively used by humans: When you start from a source node s and drive to somewhere 'far away', you will leave



Routing in Road Networks with Transit Nodes, Figure 1
Finding the optimal travel time between two points (*flags*) somewhere between Saarbrücken and Karlsruhe amounts to retrieving the 2 × 4 *access nodes* (*diamonds*), performing 16 table lookups between all pairs of access nodes, and checking that the two disks defining the *locality filter* do not overlap. *Transit nodes* that do not belong to the access node sets of the selected source and target nodes are drawn as small squares. The figure draws the levels of the highway hierarchy using colors gray, red, blue, and green for levels 0–1, 2, 3, and 4, respectively

your current location via one of only a few 'important' traffic junctions, called (forward) *access nodes* $\overrightarrow{A}(s)$. An analogous argument applies to the target t, i. e., the target is reached from one of only a few backward access nodes $\overleftarrow{A}(t)$. Moreover, the union of all forward and backward access nodes of all nodes, called *transit-node set* $\mathcal{T}$, is rather small. The two observations imply that for each node the distances to/from its forward/backward access nodes and for each transit-node pair (u, v), the distance between u and v can be stored. For given source and target nodes s and t, the length of the shortest path that passes at least one transit node is given by

$$d_{\mathcal{T}}(s,t) = \min\{d(s,u) + d(u,v) + d(v,t) \mid u \in \overrightarrow{A}(s), v \in \overleftarrow{A}(t)\} .$$

Note that all involved distances $d(s, u)$, $d(u, v)$, and $d(v, t)$ can be directly looked up in the precomputed data structures. As a final ingredient, a *locality filter* $L : V \times V \to \{\mathsf{true}, \mathsf{false}\}$ is needed that decides whether given nodes s and t are too close to travel via a transit node. L has to fulfill the property that $\neg L(s, t)$ implies that $d(s, t) = d_{\mathcal{T}}(s, t)$. Note that in general the converse need not hold since this might hinder an efficient realization of the locality filter. Thus, *false positives*, i. e., "$L(s, t) \wedge d(s, t) = d_{\mathcal{T}}(s, t)$", may occur.

The following algorithm can be used to compute $d(s, t)$:

If $\neg L(s, t)$, then compute and return $d_{\mathcal{T}}(s, t)$;
else, use any other routing algorithm.

Figure 1 gives an example. Knowing the length of the shortest path, a complete description of it can be efficiently derived using iterative table lookups and precomputed representations of paths between transit nodes. Provided that the above observations hold and that the percentage of false positives is low, the above algorithm is very efficient since a large fraction of all queries can be handled in line 1, $d_{\mathcal{T}}(s, t)$ can be computed using only a few table lookups, and source and target of the remaining queries in line 2 are quite close. Indeed, the remaining queries can be further accelerated by introducing a secondary layer of transit-node routing, based on a set of *secondary transit nodes* $\mathcal{T}_2 \supset \mathcal{T}$. Here, it is not necessary to compute and store a complete $\mathcal{T}_2 \times \mathcal{T}_2$ distance table, but it is sufficient to store only distances $\{d(u, v) \mid u, v \in \mathcal{T}_2 \wedge d(u, v) \neq d_{\mathcal{T}}(s, t)\}$, i. e., distances that cannot be obtained using the primary layer. Analogously, further layers can be added.

Routing in Road Networks with Transit Nodes, Table 1

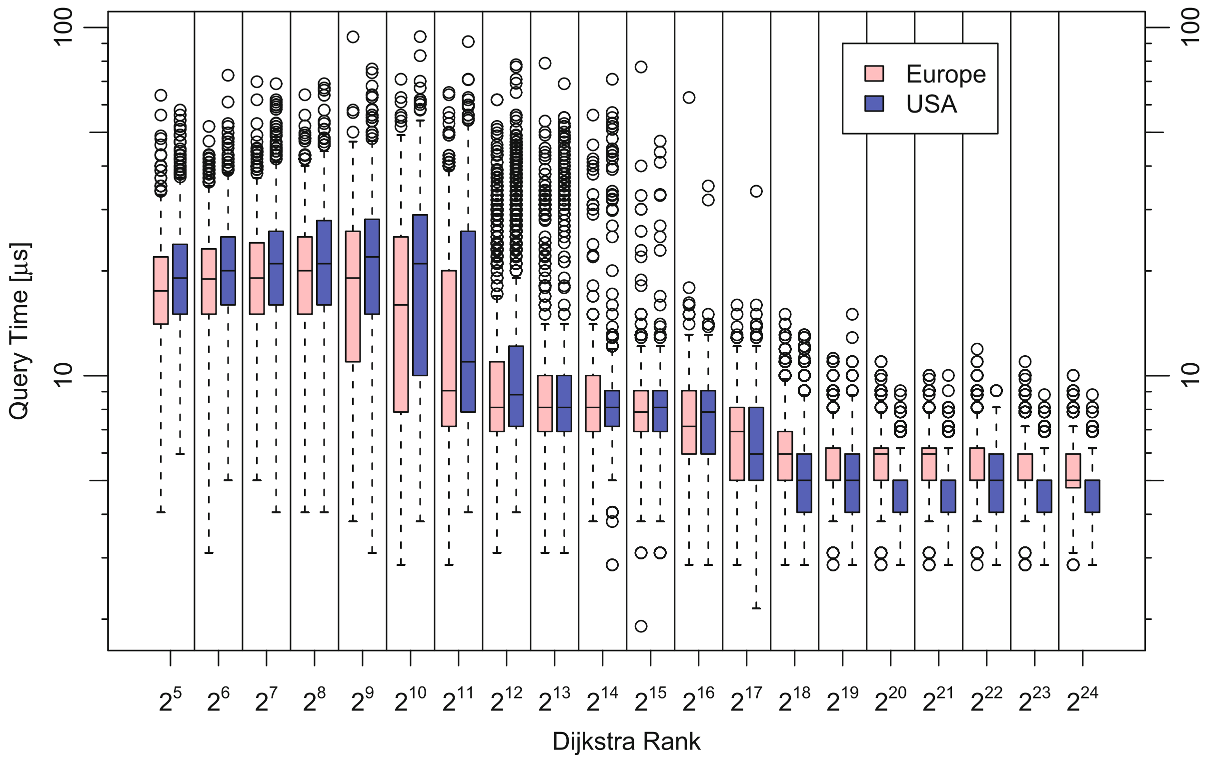
Statistics on preprocessing. The size of transit-node sets, the number of entries in distance tables, and the average number of access nodes to the respective layer are given; furthermore, the space overhead and the preprocessing time

	layer 1		layer 2			layer 3			
	$\lvert\mathcal{T}\rvert$	$\lvert A\rvert$ avg.	$\lvert\mathcal{T}_2\rvert$	$\lvert\text{table}_2\rvert$ $[\times 10^6]$	$\lvert A_2\rvert$ avg.	$\lvert\mathcal{T}_3\rvert$	$\lvert\text{table}_3\rvert$ $[\times 10^6]$	space [B/node]	time [h]
Europe	11 293	9.9	323 356	130	4.1	2 954 721	119	251	2:44
USA	10 674	5.7	485 410	204	4.2	3 855 407	173	244	3:25

Routing in Road Networks with Transit Nodes, Table 2

Performance of transit-node routing with respect to 10 000 000 random queries. The column for layer *i* specifies which fraction of the queries is correctly answered using only information available at layers $\leq i$. Each box spreads from the lower to the upper quartile and contains the median, the whiskers extend to the minimum and maximum value omitting outliers, which are plotted individually

	#nodes	#edges	layer 1	layer 2	layer 3	query
Europe	18 029 721	42 199 587	99.74%	99.9984%	99.99981%	5.6 μs
USA	24 278 285	58 213 192	99.89%	99.9986%	99.99986%	4.9 μs



Routing in Road Networks with Transit Nodes, Figure 2

Query time distribution as a function of Dijkstra rank–the number of iterations Dijkstra's algorithm would need to solve this instance. The distributions are represented as box-and-whisker plots: each box spreads from the lower to the upper quartile and contains the median, the whiskers extended to the minimum and maximum value omitting, which are plotted individually

There are two different implementations: one is based on a simple geometric grid and one on *highway hierarchies*, the fastest previous approach [5,6]. A highway hierarchy consists of a sequence of levels (Fig. 1), where level $i+1$ is constructed from level i by bypassing low-degree nodes and removing edges that never appear far away from the source or target of a shortest path. Interestingly, these levels are geometrically decreasing in size and otherwise similar to each other. The highest level contains the most 'important' nodes and becomes the primary transit-node set. The nodes of lower levels are used to form the transit-node sets of subordinated layers.

Applications

Apart from the most obvious applications in car navigation systems and server-based route planning systems, transit-node routing can be applied to several other fields, for instance to massive traffic simulations and to various optimization problems in logistics.

Open Problems

It is an open question whether one can find better transit-node sets or a better locality filter so that the performance can be further improved. It is also not clear if transit-node routing can be successfully applied to other graph types than road networks. In this context, it would be desirable to derive some theoretical guarantees that apply to any graph that fulfills certain properties. For some practical applications, a *dynamic* version of transit-node routing would be required in order to deal with time-dependent networks or unexpected edge weight changes caused, for example, by traffic jams. The latter scenario can be handled by a related approach [8], which is, however, considerably slower than transit-node routing.

Experimental Results

Experiments were performed on road networks of Western Europe and the USA using a cost function that solely takes expected travel time into account. The results exhibit various tradeoffs between average query time (5 µs to 63 µs for the USA), preprocessing time (59 min to 1200 min), and storage overhead (21 bytes/node to 244 bytes/node). For the variant that uses three layers and is tuned for best query times, Tables 1 and 2 show statistics on the preprocessing and the query performance, respectively. The average query times of about 5 µs are six orders of magnitude faster than Dijkstra's algorithm. In addition, Fig. 2 gives for each rank r on the x-axis a distribution for 1 000 queries with random starting point s and the target node t for which Dijkstra's algorithm would need r iterations to find it. The three layers of transit-node routing with small transition zones in between can be recognized: for large ranks, it is sufficient to access only the primary layer yielding query times of about 5 µs, for smaller ranks, additional layers have to be accessed resulting in median query times of up to 20 µs.

Data Sets

The European road network has been provided by the company PTV AG, the US network has been obtained from the TIGER/Line Files [9]. Both graphs have also been used in the 9th DIMACS Implementation Challenge on Shortest Paths [1].

URL to Code

The source code might be published at some point in the future at http://algo2.iti.uka.de/schultes/hwy/.

Cross References

- ► All Pairs Shortest Paths in Sparse Graphs
- ► All Pairs Shortest Paths via Matrix Multiplication
- ► Decremental All-Pairs Shortest Paths
- ► Engineering Algorithms for Large Network Applications
- ► Fully Dynamic All Pairs Shortest Paths
- ► Geographic Routing
- ► Implementation Challenge for Shortest Paths
- ► Shortest Paths Approaches for Timetable Information
- ► Shortest Paths in Planar Graphs with Negative Weight Edges
- ► Single-Source Shortest Paths

Recommended Reading

1. 9th DIMACS Implementation Challenge: Shortest Paths. http://www.dis.uniroma1.it/~challenge9/ (2006)
2. Bast, H., Funke, S., Matijevic, D., Sanders, P., Schultes, D.: In transit to constant time shortest-path queries in road networks. In: Workshop on Algorithm Engineering and Experiments, 2007, pp. 46–59
3. Bast, H., Funke, S., Sanders, P., Schultes, D.: Fast routing in road networks with transit nodes. Science **316**(5824), 566 (2007)
4. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numer. Math. **1** 269–271 (1959)
5. Sanders, P., Schultes, D.: Highway hierarchies hasten exact shortest path queries. In: 13th European Symposium on Algorithms. LNCS, vol. 3669, pp. 568–579. Springer, Berlin (2005)
6. Sanders, P., Schultes, D.: Engineering highway hierarchies. In: 14th European Symposium on Algorithms. LNCS, vol. 4168, pp. 804–816. Springer, Berlin (2006)
7. Sanders, P., Schultes, D.: Engineering fast route planning algorithms. In: 6th Workshop on Experimental Algorithms. LNCS, vol. 4525, pp. 23–36. Springer, Berlin (2007)
8. Schultes, D., Sanders, P.: Dynamic highway-node routing. In: 6th Workshop on Experimental Algorithms. LNCS, vol. 4525, pp. 66–79. Springer, Berlin (2007)
9. U.S. Census Bureau, Washington, DC: UA Census 2000 TIGER/Line Files. http://www.census.gov/geo/www/tiger/tigerua/ua_tgr2k.html (2002)

R-Trees

2004; Arge, de Berg, Haverkort, Yi

KE YI
Hong Kong University of Science and Technology,
Hong Kong, China

Keywords and Synonyms

R-Trees; Spatial databases; External memory data structures

Problem Definition

Problem Statement and the I/O Model

Let S be a set of N axis-parallel hypercubes in $\mathbb{R}^d$. A very basic operation in a spatial database is to answer *window queries* on the set S. A window query Q is also an axis-parallel hypercube in $\mathbb{R}^d$ that asks us to return all hypercubes in S that intersect Q. Since the set S is typically huge in a large spatial database, the goal is to design a *disk-based*, or *external memory* data structure (often called an *index* in the database literature) such that these window queries can be answered efficiently. In addition, given S, the data structure should be constructed efficiently, and should be able to support insertions and deletions of objects.

When external memory data structures are concerned, the standard *external memory model* [2], a.k.a. the *I/O model*, is often used as the model of computation. In this model, the machine consists of an infinite-size external memory (disk) and a main memory of size M. A block of B consecutive elements can be transferred between main memory and disk in one *I/O operation* (or simply *I/O*). An external memory data structure is a structure that is stored on disk in blocks, but computation can only occur on elements in main memory, so any operation (e. g. query, update, and construction) on the data structure must be performed using a number I/Os, which is the measure for the complexity of the operation.

R-Trees

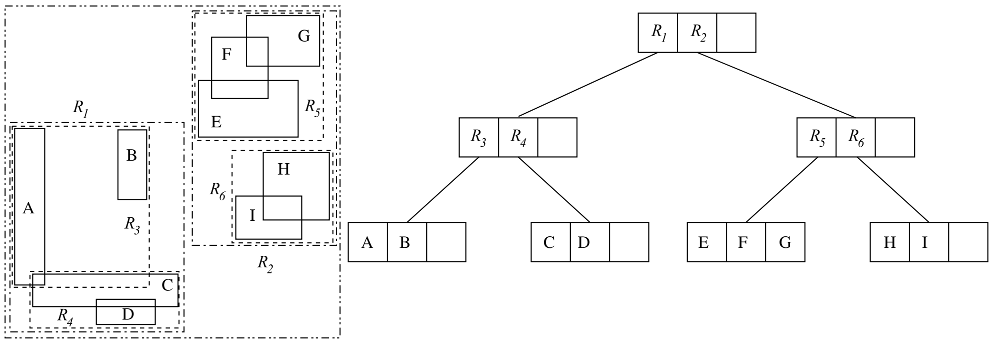
The *R-tree*, first proposed by Guttman [9], is a multi-way tree $\mathcal{T}$, very similar to a *B-tree*, that is used to store the set S such that a window query can be answered efficiently. Each node of $\mathcal{T}$ fits in one disk block. The hypercubes of S are stored only in the leaves of $\mathcal{T}$. All leaves of $\mathcal{T}$ are on the same level, and each stores $\Theta(B)$ hypercubes from S; while each internal node, except the root, has a fan-out of $\Theta(B)$. The root of $\mathcal{T}$ may have a fan-out as small as 2. For any node $u \in \mathcal{T}$, let $R(u)$ be the smallest axis-parallel hypercube, called the *minimal bounding box*, that encloses all the hypercubes stored below u. At each internal node $v \in \mathcal{T}$, whose children are denoted $v_1, \ldots, v_k$, the bounding box $R(v_i)$ is stored along with the pointer to v_i for $i = 1, \ldots, k$. Note that these bounding boxes may overlap. Please see Fig. 1 for an example of an R-tree in two dimensions.

For a window query Q, the query answering process starts from the root of $\mathcal{T}$ and visits all nodes u for which $R(u)$ intersects Q. When reaching a leaf v, it checks each hypercube stored at v to decide if it should be reported. The correctness of the algorithm is obvious, and the efficiency (the number of I/Os) is determined by the number of nodes visited.

Any R-tree occupies a linear number $O(N/B)$ disk blocks, but different R-trees might have different query, update, and construction costs. When analyzing the query complexity of window queries, the output size T is also used, in addition to N, M, and B.

Key Results

Although the structure of an R-tree is restricted, there is much freedom in grouping the hypercubes into leaves and grouping subtrees into bigger subtrees. Different grouping strategies result in different variants of R-trees. Most of the existing R-trees use various heuristics to group together hypercubes that are "close" spatially, so that a window query will not visit too many unnecessary nodes. Generally speaking, there are two ways to build an R-tree: repeated insertion and bulk-loading. The former type of algorithms include the original R-tree [9], the R$^+$-tree [15], the R*-tree [6], etc. These algorithms use $O(\log_B N)$ I/Os to insert an object and hence $O(N \log_B N)$ I/Os to build the R-tree on S, which is not scalable for large N. When the set S is known in advance, it is much more efficient to bulk-load the entire R-tree at once. Many bulk-loading algorithms have been proposed, e. g. [7,8,10,11,13]. Most of these algorithms build the R-tree with $O(N/B \log_{M/B} N/B)$ I/Os (the number of I/Os needed to sort N elements), and they typically result in better R-trees than those obtained by repeated insertion. During the past decades, there have been a large number of works devoted to R-trees from the database community, and the list here is by no means complete. The reader is referred to the book by Manolopoulos et al. [14] for an excellent survey on this subject in the database literature. However, no R-tree variant mentioned above has a guarantee on the query complexity; in fact, Arge et al. [3] constructed an example showing that some of the most popular R-trees may have to visit all the nodes without reporting a single result.



R-Trees, Figure 1
An R-tree example in two dimensions

From the theoretical perspective, the following are the two main results concerning the worst-case query complexity of R-trees.

Theorem 1 **([1,12])** *There is a set of N points in* $\mathbb{R}^d$*, such that for any R-tree* $\mathcal{T}$ *built on these points, there exists an empty window query for which the query algorithm has to visit* $\Omega((N/B)^{1-1/d})$ *nodes of* $\mathcal{T}$*.*

The *priority R-tree*, proposed by Arge et al. [3], matches the above lower bound.

Theorem 2 **([3])** *For any set S of N axis-parallel hypercubes in* $\mathbb{R}^d$*, the priority R-tree answers a window query with* $O((N/B)^{1-1/d} + T/B)$ *I/Os. It can be constructed with* $O(N/B \log_{M/B} N/B)$ *I/Os.*

It is also reported that the priority R-tree performs well in practice, too [3]. However, it is not known how to update it efficiently while preserving the worst-case bound. The logarithmic method was used to support insertions and deletions [3] but the resulted structure is no longer an R-tree.

Note that the lower bound in Theorem 1 only holds for R-trees. If the data structure is not restricted to R-trees, better query bounds can be obtained for the window-query problem; see e. g. [4].

Applications

R-trees have been used widely in practice due to its simplicity, the ability to store spatial objects of various shapes, and to answer various queries. The areas of applications span from geographical information systems (GIS), computer-aided design, computer vision, and robotics. When the objects are not axis-parallel hypercubes, they are often approximated by their minimal bounding boxes, and the R-tree is then built on these bounding boxes. To answer a window query, first the R-tree is used to locate all the intersecting bounding boxes, followed by a filtering step that checks the objects exactly. The R-tree can also be used to support other kinds of queries, for example aggregation queries, nearest-neighbors, etc. In aggregation queries, each object o in S is associated with a weight $w(o) \in \mathbb{R}$, and the goal is to compute $\sum w(o)$ where the sum is taken over all objects that intersect the query range Q. The query algorithm is same as before, except that in addition it keeps running sum while traversing the R-tree, and may skip an entire subtree rooted at some u if $R(u)$ is completely contained in Q. To find the nearest neighbor of a query point q, a priority queue is maintained, which stores all the nodes u that might contain an object that is closer to the current nearest neighbor found so far. The priority of u in the queue is the distance between q and $R(u)$. The search terminates when the current nearest neighbor is closer than the top element in the priority queue. However, no worst-case guarantees are known for R-trees answering these other types of queries, although they tend to perform well in practice.

Open Problems

Several interesting problems remain open with respect to R-trees. Some of them are listed here.

- Is it possible to design an R-tree with the optimal query bound $O((N/B)^{1-1/d} + T/B)$ that can also be efficiently updated? Or prove a lower bound on the update cost for such an R-tree.
- Is there an R-tree that supports aggregation queries for axis-parallel hypercubes in $O((N/B)^{1-1/d})$ I/Os? This would be optimal because the lower bound of Theorem 1 also holds for aggregation queries on R-trees. Note that, however, no sub-linear worst-case bound exists for nearest-neighbor queries, since it is not difficult to design a worst-case example for which the distance

between the query point q and any bounding box is smaller than the distance between q and its true nearest neighbor.

- When the window query Q shrinks to a point, that is, the query asks for all hypercubes in S that contain the query point, the problem is often referred to as *stabbing queries* or *point enclosure queries*. The lower bound of Theorem 1 does not hold for this special case; while a lower bound of $\Omega(\log_2 N + T/B)$ was proved in [5], which holds in the strong *indexability* model. It is intriguing to find out the true complexity for stabbing queries using R-trees, which is between $\Omega(\log_2 N + T/B)$ and $O((N/B)^{1-1/d} + T/B)$.

Experimental Results

Nearly all studies on R-trees include experimental evaluations, mostly in two dimensions. Reportedly the Hilbert R-tree [10,11] has been shown to have good query performance while being easy to construct. The R*-tree's insertion algorithm [6] has often been used for updating the R-tree. Please refer to the book by Manolopoulos et al. [14] for more discussions on the practical performance of R-trees.

Data Sets

Besides some synthetic data sets, the TIGER/Line data (http://www.census.gov/geo/www/tiger/) from the US Census Bureau has been frequently used as real-world data to test R-trees. The R-tree portal (http://www.rtreeportal.org/) also contains many interesting data sets.

URL to Code

Code for many R-tree variants is available at the R-tree portal (http://www.rtreeportal.org/). The code for the priority R-tree is available at http://www.cse.ust.hk/~yike/prtree/.

Cross References

▶ B-trees
▶ External Sorting and Permuting
▶ I/O-model

Recommended Reading

1. Agarwal, P.K., de Berg, M., Gudmundsson, J., Hammar, M., Haverkort, H.J.: Box-trees and R-trees with near-optimal query time. Discret. Comput. Geom. **28**, 291–312 (2002)
2. Aggarwal, A., Vitter, J.S.: The input/output complexity of sorting and related problems. In: Communications of the ACM, vol. 31, pp. 1116–1127 (1988)
3. Arge, L., de Berg, M., Haverkort, H.J., Yi, K.: The priority R-tree: A practically efficient and worst-case optimal R-tree. In: Proc. SIGMOD International Conference on Management of Data, 2004, pp. 347–358
4. Arge, L., Samoladas, V., Vitter, J.S.: On two-dimensional indexability and optimal range search indexing. In: Proc. ACM Symposium on Principles of Database Systems, 1999, pp. 346–357
5. Arge, L., Samoladas, V., Yi, K.: Optimal external memory planar point enclosure. In: Proc. European Symposium on Algorithms, 2004
6. Beckmann, N., Kriegel, H.-P., Schneider, R., Seeger, B.: The R*-tree: An efficient and robust access method for points and rectangles. In: Proc. SIGMOD International Conference on Management of Data, 1990, pp. 322–331
7. DeWitt, D.J., Kabra, N., Luo, J., Patel, J.M., Yu, J.-B.: Client-server paradise. In: Proc. International Conference on Very Large Databases, 1994, pp. 558–569
8. García, Y.J., López, M.A., Leutenegger, S.T.: A greedy algorithm for bulk loading R-trees. In: Proc. 6th ACM Symposium on Advances in GIS, 1998, pp. 163–164
9. Guttman, A.: R-trees: A dynamic index structure for spatial searching. In: Proc. SIGMOD International Conference on Management of Data, 1984, pp. 47–57
10. Kamel, I., Faloutsos, C.: On packing R-trees. In: Proc. International Conference on Information and Knowledge Management, 1993, pp. 490–499
11. Kamel, I., Faloutsos, C.: Hilbert R-tree: An improved R-tree using fractals. In: Proc. International Conference on Very Large Databases, 1994, pp. 500–509
12. Kanth, K.V.R., Singh, A.K.: Optimal dynamic range searching in non-replicating index structures. In: Proc. International Conference on Database Theory. LNCS, vol. 1540, pp. 257–276 (1999)
13. Leutenegger, S.T., Lopez, M.A., Edington, J.: STR: A simple and efficient algorithm for R-tree packing. In: Proc. 13th IEEE International Conference on Data Engineering, 1997, pp. 497–506
14. Manolopoulos, Y., Nanopoulos, A., Papadopoulos, A.N., Theodoridis, Y.: R-trees: Theory and Applications. Springer, London (2005)
15. Sellis, T., Roussopoulos, N., Faloutsos, C.: The R$^+$-tree: A dynamic index for multi-dimensional objects. In: Proc. International Conference on Very Large Databases, 1987, pp. 507–518

Runs

▶ Squares and Repetitions

S

Schedulers for Optimistic Rate Based Flow Control

2005; Fatourou, Mavronicolas, Spirakis

PANAGIOTA FATOUROU
Department of Computer Science,
University of Ioannina, Ioannina, Greece

Keywords and Synonyms

Rate allocation; Rate adjustment; Bandwidth allocation

Problem Definition

The problem concerns the design of efficient rate-based flow control algorithms for virtual-circuit communication networks where a connection is established by allocating a fixed path, called *session*, between the source and the destination. Rate-based flow-control algorithms repeatedly adjust the transmission rates of different sessions in an end-to-end manner with primary objectives to optimize the network utilization and achieve some kind of fairness in sharing bandwidth between different sessions.

A widely-accepted fairness criterion for flow-control is *max-min fairness* which requires that the rate of a session can be increased only if this increase does not cause a decrease to any other session with smaller or equal rate. Once max-min fairness has been achieved, no session rate can be increased any further without violating the above condition or exceeding the bandwidth *capacity* of some link. Call *max-min* rates the session rates when max-min fairness has been reached.

Rate-based flow control algorithms perform rate adjustments through a sequence of *operations* in a way that the capacities of network links are never exceeded. Some of these algorithms, called *conservative* [3,6,10,11,12], employ operations that gradually increase session rates until they converge to the max-min rates without ever performing any rate decreases. On the other hand, *optimistic* algorithms, introduced more recently by Afek, Mansour, and Ostfeld [1], allow for decreases, so that a session's rate may be intermediately be larger than its final max-min rate.

Optimistic algorithms [1,7] employ a specific rate adjustment operation, called *update operation* (introduced in [1]). The goal of an `update` operation is to achieve fairness among a set of neighboring sessions and optimize the network utilization in a local basis. More specifically, an `update` operation calculates an increase for the rate of a particular session (the *updated* session) for each link the session traverses. The calculated increase on a particular link is the maximum possible that respects the max-min fairness condition between the sessions traversing the link; that is, this increase should not cause a decrease to the rate of any other session traversing the link with smaller rate than the rate of the updated session after the increase. Once the maximum increase on each link has been calculated the minimum among them is applied to the session's rate (let e be the link for which the minimum increase has been calculated). This causes the decrease of the rates of those sessions traversing e which had larger rates than the increased rate of the updated session to the new rate. Moreover, the `update` operation guarantees that all the capacity of link e is allocated to the sessions traversing it (so the bandwidth of this link is fully utilized).

One important performance parameter of a rate-based flow control algorithm is its *locality* which is characterized by the amount of knowledge the algorithm requires to decide which session's rate to update next. *Oblivious* algorithms do not assume any knowledge of the network topology or the current session rates. *Partially oblivious* algorithms have access to session rates but they are unaware of the network topology, while *non-oblivious* algorithms require full knowledge of both the network topology and the session rates. Another crucial performance parameter of rate-based flow control algorithms is the *convergence complexity* measured as the maximum number of rate-adjustment operations performed in any execution until max-min fairness is achieved.

Key Results

Fatourou, Mavronicolas and Spirakis [7] have studied the convergence complexity of optimistic rate-based flow control algorithms under varying degrees of locality. More specifically, they have proved lower and upper bounds on the convergence complexity of oblivious, partially-oblivious and non-oblivious algorithms. These bounds are expressed in terms of n the number of sessions laid out on the network.

Theorem 1 (Lower Bound for Oblivious Algorithms, Fatourou, Mavronicolas and Spirakis [7]) *Any optimistic, oblivious, rate-based flow control algorithm requires $\Omega(n^2)$ `update` operations to compute the max-min rates.*

Fatourou, Mavronicolas and Spirakis [7] have presented algorithm `RoundRobin`, which applies `update` operations to sessions in a round robin order. Obviously, `RoundRobin` is oblivious. It has been proved [7] that the convergence complexity of `RoundRobin` is $O(n^2)$. This shows that the lower bound for oblivious algorithms is tight.

Theorem 2 (Upper Bound for Oblivious Algorithms, Fatourou, Mavronicolas and Spirakis [7]) *`RoundRobin` computes the max-min rates after performing $O(n^2)$ `update` operations.*

`RoundRobin` belongs to a class of oblivious algorithms, called `Epoch` [7]. Each algorithm of this class repeatedly chooses some permutation of all session indices and applies `update` operations on the sessions in the order determined by this permutation. This is performed n times. Clearly, `Epoch` is a class of oblivious algorithms. It has been proved [7] that each of the algorithms in this class has convergence complexity $O(n^2)$.

Another oblivious algorithm, called `Arbitrary`, has been presented in [1]. The algorithm works in a very simple way by choosing the next session to be updated in an arbitrary way, but it requires an exponential number of `update` operations to compute the max-min rates.

Fatourou, Mavronicolas and Spirakis [7] have proved that partially-oblivious algorithms do not achieve better convergence complexity than oblivious algorithms despite the knowledge they employ.

Theorem 3 (Lower Bound for Partially Oblivious Algorithms, Fatourou, Mavronicolas and Spirakis [7]) *Any optimistic, partially oblivious, rate-based flow control algorithm requires $\Omega(n^2)$ `update` operations to compute the max-min rates.*

Afek, Mansour and Ostfeld [1] have presented a partially oblivious algorithm, called `GlobalMin`. The algorithm chooses as the session to update next the one with the minimum rate among all sessions. The convergence complexity of `GlobalMin` is $O(n^2)$ [1]. This shows that the lower bound for partially-oblivious algorithms is tight.

Theorem 4 (Upper Bound for Partially Oblivious algorithms, Afek, Mansour and Ostfeld [1]) *`GlobalMin` computes the max-min rates after performing $O(n^2)$ `update` operations.*

Another partially-oblivious algorithm, called `LocalMin`, is also presented in [1]. The algorithm chooses to schedule next a session which has a minimum rate among all the sessions that share a link with it. `LocalMin` has time complexity $O(n^2)$.

Fatourou, Mavronicolas and Spirakis [7] have presented a non-oblivious algorithm, called `Linear`, that exhibits linear convergence complexity. `Linear` follows the classical idea [3,12] of selecting as the next updated session one of the sessions that traverse the most congested link in the network. To discover such a session, `Linear` requires knowledge of the network topology and the session rates.

Theorem 5 (Upper Bound for Non-Oblivious Algorithms, Fatourou, Mavronicolas and Spirakis [7]) *`Linear` computes the max-min rates after performing $O(n)$ `update` operations.*

The convergence complexity of `Linear` is optimal, since n rate adjustments must be performed in any execution of an optimistic rate-based flow control algorithm (assuming that the initial session rates are zero). However, this comes at a remarkable cost in locality which makes `Linear` impractical.

Applications

Flow control is the dominant technique used in most communication networks for preventing data traffic congestion when the externally injected transmission load is larger than what can be handled even with optimal routing. Flow control is also used to ensure high network utilization and fairness among the different connections. Examples of networking technologies where flow control techniques have been extensively employed to achieve these goals are TCP streams [5] and ATM networks [4]. An overview of flow control in practice is provided in [3].

The idea of controlling the rate of a traffic source originates back to the data networking protocols of the ANSI Frame Relay Standard. Rate-based flow control is considered attractive due to its simplicity and its low hardware requirements. It has been chosen by the ATM Forum on Traffic Management as the best suited technique for the goals of ABR service [4].

A substantial amount of research work has been devoted in past to conservative flow control algorithms [3,6,10,11,12]. The optimistic framework has been introduced much later by Afek et al. [1] as a more suitable approach for real dynamic networks where decreases of session rates may be necessary (e. g., for accommodating the arrival of new sessions). The algorithms presented in [7] improve upon the original algorithms proposed in [1] in terms of either convergence complexity, or locality, or both. Moreover, they identify that certain classical scheduling techniques, such as round-robin [11], or adjusting the rates of sessions traversing one of the most congested links [3,12] can be efficient under the optimistic framework. The first general lower bounds on the convergence complexity of rate-based flow control algorithms are also presented in [7].

The performance of optimistic algorithms has been theoretically analyzed in terms of an abstraction, namely the `update` operation, which has been designed to address most of the intricacies encountered by rate-based flow control algorithms. However, the `update` operation masks low-level implementation details, while it may incur non-trivial, local computations on the switches of the network. Fatourou, Mavronicolas and Spirakis [9] have studied the impact on the efficiency of optimistic algorithms of local computations required at network switches in order to implement the `update` operation, and proposed a distributed scheme that implements a broad class of such algorithms. On a different avenue, Afek, Mansour and Ostfeld [2] have proposed a simple flow control scheme, called `Phantom`, which employs the idea of considering an imaginary session on each link [10,12], and they have discussed how `Phantom` can be applied to ATM networks and networks of TCP routers.

A broad class of modern distributed applications (e. g., remote video, multimedia conferencing, data visualization, virtual reality, etc.) exhibit highly differing bandwidth requirements and need some kind of quality of service guarantees. To efficiently support a wide diversity of applications sharing available bandwidth, a lot of research work has been devoted on incorporating priority schemes on current networking technologies. Priorities offer a basis for modeling the diverse resource requirements of modern distributed applications, and they have been used to accommodate the needs of network management policies, traffic levels, or pricing. The first efforts for embedding priority issues into max-min fair, rate-based flow control were performed in [10,12]. An extension of the classical theory of max-min fair, rate-based flow control to accommodate priorities of different sessions has been presented in [8]. (A number of other papers addressing similar generalizations of max-min fairness to account for priorities and utility have been presented after the original publication of [8].)

Many modern applications are not based solely on point-to-point communication but they rather require multipoint-to-multipoint transmissions. A max-min fair rate-based flow control algorithm for multicast networks is presented in [14]. Max-min fair allocation of bandwidth in wireless adhoc networks is studied in [15].

Open Problems

The research work on optimistic, rate-based flow control algorithms leaves open several interesting questions. The convergence complexity of the proposed optimistic algorithms has been analyzed only for a static set of sessions laid out on the network. It would be interesting to evaluate these algorithms under a dynamic network setting, and possibly extend the techniques they employ to efficiently accommodate arriving and departing sessions.

Although max-min fairness has emerged as the most frequently praised fairness criterion for flow control algorithms, achieving it might be expensive in highly dynamic situations. Afek et al. [1] have proposed a modified version of the `update` operation, called *approximate* `update`, which applies an increase to some session only if it is larger than some quantity $\delta > 0$. An *approximate* optimistic algorithm uses the approximate `update` operation and terminates if no session rate can be increased by more than δ. Obviously such an algorithm does not necessarily reach max-min fairness. It has been proved [1] that for some network topologies every approximate optimistic algorithm may converge to session rates that are away from their max-min counterparts by an exponential factor. The consideration of other versions of `update` operation or different termination conditions might lead to better max-min fairness approximations and deserves more study; different choices may also significantly impact the convergence complexity of approximate optimistic algorithms. It would be also interesting to derive trade-off results between the convergence complexity of such algorithms and the distance of the terminating rates they achieve to the max-min rates.

Fairness formulations that naturally approximate the max-min condition have been proposed by Kleinberg et al. [13] as suitable fairness criteria for certain routing and load balancing applications. Studying these formulations under the rate-based flow control setting is an interesting open problem.

Cross References

► Multicommodity Flow, Well-linked Terminals and Routing Problems

Recommended Reading

1. Afek, Y., Mansour, Y., Ostfeld, Z.: Convergence complexity of optimistic rate based flow control algorithms. J. Algorithms **30**(1), 106–143 (1999)
2. Afek, Y., Mansour, Y., Ostfeld, Z.: Phantom: a simple and effective flow control scheme. Comput. Netw. **32**(3), 277–305 (2000)
3. Bertsekas, D.P., Gallager, R.G.: Data Networks, 2nd edn. Prentice Hall, Englewood Cliffs (1992)
4. Bonomi, F., Fendick, K.: The Rate-Based Flow Control for Available Bit Rate ATM Service. IEEE/ACM Trans. Netw. **9**(2), 25–39 (1995)
5. Brakmo, L.S., Peterson, L.: TCP Vegas: End-to-end Congestion Avoidance on a Global Internet. IEEE J. Sel. Areas Commun. **13**(8), 1465–1480 (1995)
6. Charny, A.: An algorithm for rate-allocation in a packet-switching network with feedback. Technical Report MIT/LCS/TR-601, Massachusetts Institute of Technology, April 1994
7. Fatourou, P., Mavronicolas, M., Spirakis, P.: Efficiency of oblivious versus non-oblivious schedulers for optimistic, rate-based flow control. SIAM J. Comput. **34**(5), 1216–1252 (2005)
8. Fatourou, P., Mavronicolas, M., Spirakis, P.: Max-min fair flow control sensitive to priorities. J. Interconnect. Netw. **6**(2), 85–114 (2005) (also in Proceedings of the 2nd International Conference on Principles of Distributed Computing, pp. 45–59 (1998)
9. Fatourou, P., Mavronicolas, M., Spirakis, P.: The global efficiency of distributed, rate-based flow control algorithms. In: Proceedings of the 5th Colloqium on Structural Information and Communication Complexity, pp. 244–258, June 1998
10. Gafni, E., Bertsekas, D.: Dynamic control of session input rates in communication networks. IEEE Trans. Autom. Control **29**(11), 1009–1016 (1984)
11. Hahne, E.: Round Robin Scheduling for Max-min Fairness in Data Networks. IEEE J. Sel. Areas Commun. **9**(7), 1024–1039 (1991)
12. Jaffe, J.: Bottleneck Flow Control. IEEE Trans. Commun. **29**(7), 954–962 (1981)
13. Kleinberg, J., Rabani, Y., Tardos, É.: Fairness in routing and load balancing. In: Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science, pp. 568–578, October 1999
14. Sarkar, S., Tassiulas, L.: Fair distributed congestion control in multirate multicast networks. IEEE/ACM Trans. Netw. **13**(1), 121–133 (2005)
15. Tassiulas, L., Sarkar, S.: Maxmin fair scheduling in wireless adhoc networks. IEEE J. Sel. Areas Commun. **23**(1), 163–173 (2005)

Scheduling with Equipartition

2000; Edmonds

JEFF EDMONDS
York University, Toronto, ON, Canada

Keywords and Synonyms

Round Robin and Equi-partition are the same algorithm. Average Response time and Flow are basically the same measure.

Problem Definition

The task is to schedule a set of n on-line jobs on p processors. The jobs are $J = \{J_1, \ldots, J_n\}$ where *job* J_i has a *release/arrival time* r_i and a sequence of phases $\langle J_i^1, J_i^2, \ldots, J_i^{q_i} \rangle$. Each phase is represented by $\langle w_i^q, \Gamma_i^q \rangle$, where w_i^q denotes the amount of *work* and Γ_i^q is the *speedup function* specifying the rate $\Gamma_i^q(\beta)$ at which this work is executed when given β processors.

A phase of a job is said to be *fully parallelizable* if its speedup function is $\Gamma(\beta) = \beta$. It is said to be *sequential* if its speedup function is $\Gamma(\beta) = 1$.[1] A speedup function Γ is *nondecreasing* iff $\Gamma(\beta_1) \le \Gamma(\beta_2)$ whenever $\beta_1 \le \beta_2$;[2] is *sublinear* iff $\Gamma(\beta_1)/\beta_1 \ge \Gamma(\beta_2)/\beta_2$;[3] and is *strictly-sublinear* by α iff $\Gamma(\beta_2)/\Gamma(\beta_1) \le (\beta_2/\beta_1)^{1-\alpha}$.

An *s-speed scheduling algorithm* $S_s(J)$ allocates $s \times p$ processors each point in time to the jobs J in a way such that all the work completes.[4] More formally, it constructs a function $S(i,t)$ from $\{1, \ldots, n\} \times [0, \infty)$ to $[0, sp]$ giving the number of processors allocated to job J_i at time t. (A job is allowed to be allocated a non-integral number of processors.) Requiring that for all t, $\sum_{i=1}^{n} S(i,t) \le sp$ ensures that at most sp processors are allocated at any given time. Requiring that for all i, there exist $r_i = c_i^0 < c_i^1 < \cdots < c_i^{q_i}$ such that for all $1 \le q \le q_i$,

[1]Note that an odd feature of this definition is that a sequential job completes work at a rate of 1 even when absolutely no processors are allocated to it. This assumption makes things easier for the adversary and harder for any non-clairvoyant algorithm. Hence, it only makes these results stronger.

[2]A job phase with a nondecreasing speedup function executes no slower if it is allocated more processors.

[3]A measure of how efficient a job utilizes its processors is $\Gamma(\beta)/\beta$, which is the work completed by the job per time unit per processor. A sublinear speedup function is one whose efficiency does not increase with more processors. This is a reasonable assumption if in practice β_1 processors can simulate the execution of β_2 processors in a factor of at most β_2/β_1 more time.

[4]$S^s(J)$ is defined to be the scheduler with p processors of speed s. S_s and S^s are equivalent on fully parallelizable jobs and S^s is s times faster than S_s on sequential jobs.

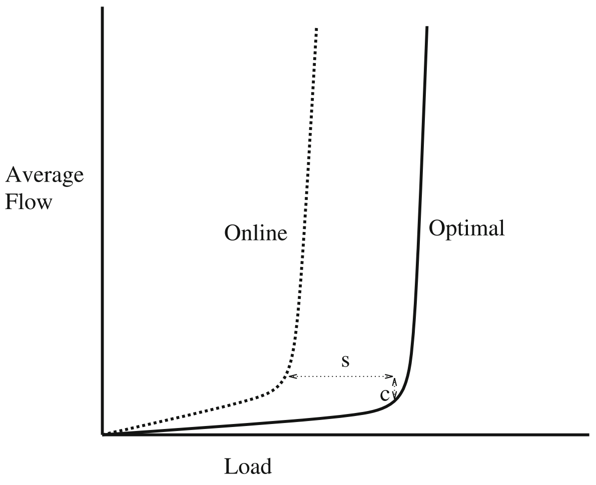
$\int_{c_i^{q-1}}^{c_i^q} \Gamma_i^q(S(i,t))\,dt = w_i^q$ ensures that before a phase of a job begins, the job must have been released and all of the previous phases of the job must have completed. The *completion* time of a job J_i, denoted c_i, is the completion time of the last phase of the job.

The goal of a scheduling algorithm is to minimize the *average response time*, $\frac{1}{n}\sum_{i\in J}(c_i - r_i)$, of the jobs or equivalently its *flow time* $S_s(J) = \sum_{i\in J}(c_i - r_i)$. An alternative formalization is to integrate over time the number of jobs n_t alive at time t, $S_s(J) = \sum_{i\in J}\int_0^\infty (J_i \text{ is alive a time } t)\delta t = \int_0^\infty n_t \delta t$.

A scheduling algorithm is said to be *on-line* if it lacks knowledge of which jobs will arrive in the future. It is said to be *non-clairvoyant* if it also lacks all knowledge about the jobs that are currently in the system, except for knowing when a job arrives and knowing when it completes.

The two examples of *non-clairvoyant* schedulers that are often used in practice are *Equi-partition* (also called *Round Robin*) and *Balance*. $EQUI_s$ is defined to be the scheduler that allocates an equal number of processors to each job that is currently alive. That is, for all i and t, if job J_i is alive at time t, then $\mathcal{EQUI}(i,t) = sp/n_t$, where n_t is the number of jobs that are alive at time t. The schedule BAL_s is defined in [8] to be the schedule that allocates all of its processors to the job that has been allocated processors for the shortest length of time. (Though no one implements Balance directly, Unix uses a multi-level feedback (MLF) queue algorithm which in a way approximates Balance).

The most obvious worst-case measure of the goodness of an online non-clairvoyant scheduling algorithm S is its *competitive ratio*. This compares the perform of the algorithm to that of the optimal scheduler. However, in many cases, the limited algorithm is unable to compete against an all knowing all powerful optimal scheduler. To compensate the algorithm S_s, it is given extra speed s. An online scheduling algorithm S is said to be *s-speed c-competitive* if: $\max_J S_s(J)/Opt(J) \le c$. For example, being s-speed 2-competitive means that the cost $S_s(J)$ of scheduler S with $s \times p$ processors on any instance J is at most twice the optimal cost for the same jobs when only given p processors.



Scheduling with Equipartition, Figure 1
To understand the motivation for this *resource augmentation analysis* [8], note that it is common for the quality of service of a system to have a *threshold property* with respect to the load that it is given. In this example, it seems that the online scheduling algorithm *S* performs reasonably well in comparison to the optimal scheduling algorithm. Despite this, one can see that the competitive ratio of *S* is huge by looking at the vertical gap between the curves when the load is near capacity. To explain why these curves are close, one must also measure the horizontal gap between curves. *S* performs at most *c* times worse than optimal, when either the load is decreased or equivalently the speed is increased by factor of *s*

Key Results

If all jobs arrive at time zero (batch), then the flow time of *EQUI* is 2-competitive on fully parallelizable jobs [10] and $(2+\sqrt{3})$-competitive on jobs with nondecreasing sublinear speedup functions [3]. (The time until the last job completes (makespan) on fully parallelizable jobs is the same for *EQUI* and *OPT*, but can be a factor of $\Theta(\log n/\log\log n)$ worse for *EQUI* if the jobs can also have sequential phases [11].) Table 1 summarizes all the results.

When the jobs have arbitrary arrival times and are fully parallelizable, the optimal schedule simply allocates all the processors to the jobs with least remaining work. This, however, requires the scheduler to know the amount of work per job. Without this knowledge, *EQUI* and *BAL* are unable to compete with the optimal and hence can do a factor of $\Omega(n/\log n)$ and $\Omega(n)$ respectively worse and no non-clairvoyant schedulers has a better competitive ratio than $\Omega(n^{1/3})$ [9,10]. Randomness improves the competitive ratio of *BAL* to $\Theta(\log n \log\log n)$ [7]. Having more (or faster) processors also helps. BAL_s achieves a $s = 1+\epsilon$ speed competitive ratio of $\frac{s}{s-1} = 1 + \frac{1}{\epsilon}$ [8].

If some of the jobs are fully parallelizable and some are sequential jobs, it is hard to believe that any non-clairvoyant scheduler, even with sp processors, can perform well. Not knowing which jobs are which, it waists too many processors on the sequential jobs. Being starved, the fully parallelizable jobs fall further and further behind and then other fully parallelizable jobs arrive which fall behind as well. For example, even the randomized version of *BAL*

Scheduling with Equipartition, Table 1
Each row represents a specific scheduler and a class $\mathcal{J}$ of job sets. Here $EQUI_s$ denotes the Equi-partition scheduler with s times as many processors and $EQUI^s$ the one with processors that are s times as fast. The graphs give examples of speedup functions from the class of those considered. The columns are for different extra resources ratios s. Each entry gives the corresponding ratio between the given scheduler and the optimal

	$s = 1$	$s = 1 + \epsilon$	$s = 2 + \epsilon$	$s = 4 + 2\epsilon$	$s = \mathcal{O}(\log p)$
Batch , , or	$[2.71, 3.74]$				
Det. Non-clair	$\Omega(n^{\frac{1}{3}})$	–			
Rand. Non-clair	$\widetilde{\Theta}(\log n)$	–			
Rand. Non-clair or	$\Omega(n^{\frac{1}{2}})$	$\Omega(\frac{1}{\epsilon})$			
BAL_s	$\Omega(n)$	$1 + \frac{1}{\epsilon}$	$\frac{2}{s}$		
BAL_s	$\Omega(s^{-1/\alpha} n)$				
$EQUI_s$, , or	$\Omega(\frac{n}{\log n})$	$\Omega(n^{1-\epsilon})$	$[1 + \frac{1}{\epsilon}, 2 + \frac{4}{\epsilon}]$	≥ 1	
$EQUI^s$, , or	$\Omega(\frac{n}{\log n})$	$\Omega(n^{1-\epsilon})$	$[\frac{2}{3}(1 + \frac{1}{\epsilon}), 2 + \frac{4}{\epsilon}]$	$[\frac{2}{s}, \frac{16}{s}]$	
$EQUI$ or	$[1.48^{1/\alpha}, 2^{1/\alpha}]$				
$EQUI'_s$ Few Preempts			$\Omega(n^{1-\epsilon})$	$\Theta(1)$	
$H\,EQUI_s$ or			$\Omega(n^{1-\epsilon})$	$\Theta(1)$	
$H\,EQUI'_s$ or	$\Omega(n)$				$\Theta(1)$

can have an arbitrarily bad competitive ratio, even when given arbitrarily fast processors.

EQUI, however, does amazingly well. $EQUI_s$ achieves a $s = 2 + \epsilon$ speed competitive ratio of $2 + \frac{4}{\epsilon}$ [1]. This was later improved to $1 + \mathcal{O}(\sqrt{s}/(s-2))$, which is better for large s [1]. The intuition is that $EQUI_s$ is able to automatically "self adjust" the number of processors wasted on the sequential jobs. As it falls behind, it has more uncompleted jobs in the system and hence allocates fewer processors to each job and hence each job utilizes the processors that it is given more efficiently. The extra processors are enough to compensate for the fact that some processors are still wasted on sequential jobs. For example, suppose the job set is such that *OPT* has ℓ_t sequential jobs and at most one fully parallelizable job alive at any point in time t. (The proof starts by proving that this is the worst case.) It may take a while for the system under $EQUI_s$ to reach a "steady state", but when it does, m_t, which denotes the number of fully parallelizable jobs it has alive at time t, converges to $\frac{\ell_t}{s-1}$. At this time, $EQUI_s$ has $\ell_t + m_t$ jobs alive and *OPT* has $\ell_t + 1$. Hence, the competitive ratio is $EQUI_s(J)/OPT(J) = (\ell_t + \frac{\ell_t}{s-1}))/(\ell_t + 1) \leq \frac{s}{s-1}$, which is $1 + \frac{1}{e}$ for $s = 1 + \epsilon$. This intuition makes it appear that speed $s = 1 + \epsilon$ is sufficient. However, unless the speed is at least 2 then the competitive ratio can be bad during the time until it reaches this steady state, [8].

More surprisingly if all the jobs are *strictly sublinear*, i. e., are not fully parallel, then *EQUI* performs competitively with no extra processors [1]. More specifically, it is shown that if all the speedup functions are no more fully parallelizable than $\Gamma(\beta) = \beta^{1-\alpha}$ than the competitive ratio is at most $2^{\frac{1}{\alpha}}$. For intuition, suppose the adversary allocates $\frac{p}{n}$ processors to each of n jobs and *EQUI* falls behind enough so that it has $2^{\frac{1}{\alpha}} n$ uncompleted jobs. Then it allocates $p/(2^{\frac{1}{\alpha}} n)$ processors to each, completing work at an overall rate of $(2^{\frac{1}{\alpha}} n)\Gamma(p/(2^{\frac{1}{\alpha}} n)) = 2 \cdot n\Gamma(p/n)$. This is a factor of 2 more than that by the adversary. Hence, as in the previous result, *EQUI* has twice the speed and so performs competitively.

The results for $EQUI_s$ can be extended further. There is a competitive $s = (8 + \epsilon)$-speed non-clairvoyant scheduler that only preempts when the number of jobs in the system goes up or down by a factor of two (in some sense $\log n$ times). There is $s = (4 + \epsilon)$-speed one that includes both sublinear and superlinear jobs. Finally, there is a $s = \mathcal{O}(\log p)$ speed one that includes both nondecreasing and gradual jobs.

The proof of these results for $EQUI_s$ require techniques that are completely new. For example, the previous results prove that their algorithm is competitive by proving that at every point in time, the number of jobs alive under their algorithm is within a constant fraction of that under the optimal schedule. This, however, is simply not true with this less restricted model. There are job sets such that for a period of time the ratio between the numbers of alive jobs under the two schedules is unbounded. Instead, a potential function is used to prove that this can only happen for a relatively short period of time.

The proof first transforms each possible input into a canonical input that as described above only has parallelizable or sequential phases. Having the number of fully parallelizable jobs alive under $EQUI_s$ at time t be much bigger than the number of sequential jobs alive at this same time is bad for $EQUI_s$ because it then has many more jobs alive then OPT and hence is currently incurring much higher costs. On the other hand, this same situation is also good for $EQUI_s$ because it means that it is allocating a larger fraction of its processors to the fully parallelizable jobs and hence is catching up to OPT. Both of these aspects of the current situation is carefully measured in a potential function $\Phi(t)$. It is proven that at each point in time, the occurred cost to $EQUI_s$ plus the gain $(\mathrm{d}\Phi(t))/(\mathrm{d}t)$ in this potential function is at most c times the costs occurred by OPT. Assuming that the potential function begins and ends at zero, the result follows.

More formally, the potential function is $\Phi(t) = F(t) + Q(t)$ where $Q(t)$ is total sequential work finished by $EQUI_s$ by time t minus the total sequential work finished by the adversary by time t. To define $F(t)$ requires some preliminary definitions. For $u \geq t$, define $m_u(t)$ $(\ell_u(t))$ to be number of fully parallelizable (sequential) phases executing under $EQUI_s$ at time u, for which $EQUI_s$ at time u has still not processed as much work as the adversary processed at time t. Let $n_u(t) = m_u(t) + \ell_u(t)$. Then $F(t) = \int_t^\infty f_u(m_u(t), \ell_u(t))\,\mathrm{d}u$, where $f_u(m, \ell) = \frac{s}{s-2} \frac{(m-\ell)(m+\ell)}{n_u}$. As the definition of the potential function suggests, the analysis is quite complicated.

Applications

In addition to being interesting results on their own, they have been powerful tools for the theoretical analysis of other on-line algorithms. For example, in [2,4] TCP was reduced to this problem and in [5], the online broadcast scheduling problem was reduced to this problem.

Open Problems

An open question is whether there is an algorithm that is competitive when given processors of speed $s = 1 + \epsilon$ (as opposed to $s = 2 + \epsilon$). There is a candidate algorithm that is part way between $EQUI_s$ and BAL_s.

Cross References

▶ Flow Time Minimization
▶ List Scheduling
▶ Load Balancing
▶ Minimum Flow Time
▶ Minimum Weighted Completion Time
▶ Online List Update
▶ Schedulers for Optimistic Rate Based Flow Control
▶ Shortest Elapsed Time First Scheduling

Recommended Reading

1. Edmonds, J.: Scheduling in the dark. Improved results: manuscript 2001. In: Theor. Comput. Sci. **235**, 109–141 (2000). In: 31st Ann. ACM Symp. on Theory of Computing, 1999
2. Edmonds, J.: On the Competitiveness of AIMD-TCP within a General Network. In: LATIN, Latin American Theoretical Informatics, vol. 2976, pp. 577–588 (2004). Submitted to Journal Theoretical Computer Science and/or Lecture Notes in Computer Science
3. Edmonds, J., Chinn, D., Brecht, T., Deng, X.: Non-clairvoyant Multiprocessor Scheduling of Jobs with Changing Execution Characteristics. In: 29th Ann. ACM Symp. on Theory of Computing, 1997, pp. 120–129. Submitted to SIAM J. Comput.
4. Edmonds, J., Datta, S., Dymond, P.: TCP is Competitive Against a Limited Adversary. In: SPAA, ACM Symp. of Parallelism in Algorithms and Achitectures, 2003, pp. 174–183
5. Edmonds, J., Pruhs, K.: Multicast pull scheduling: when fairness is fine. Algorithmica **36**, 315–330 (2003)
6. Edmonds, J., Pruhs, K.: A maiden analysis of longest wait first. In: Proc. 15th Symp. on Discrete Algorithms (SODA)
7. Kalyanasundaram, B., Pruhs, K.: Minimizing flow time nonclairvoyantly. In: Proceedings of the 38th Symposium on Foundations of Computer Science, October 1997
8. Kalyanasundaram, B., Pruhs, K.: Speed is as powerful as clairvoyance. In: Proceedings of the 36th Symposium on Foundations of Computer Science, October 1995, pp. 214–221
9. Matsumoto: Competitive Analysis of the Round Robin Algorithm. in: 3rd International Symposium on Algorithms and Computation, 1992, pp. 71–77
10. Motwani, R., Phillips, S., Torng, E.: Non-clairvoyant scheduling. Theor. Comput. Sci. **130** (Special Issue on Dynamic and On-Line Algorithms), 17–47 (1994). Preliminary Version in: Proceedings of the 4th Annual ACM-SIAM Symposium on Discrete Algorithms, 1993, pp. 422–431
11. Robert, J., Schabanel, N.: Non-Clairvoyant Batch Sets Scheduling: Fairness is Fair enough. Personal Correspondence (2007)

Scheduling with Unknown Job Sizes

▶ Multi-level Feedback Queues
▶ Shortest Elapsed Time First Scheduling

Searching

▶ Deterministic Searching on the Line

Selfish Unsplittable Flows: Algorithms for Pure Equilibria

2005; Fotakis, Kontogiannis, Spirakis

PAUL SPIRAKIS
Computer Engineering and Informatics, Research and Academic Computer Technology Institute, Patras University, Patras, Greece

Keywords and Synonyms

Atomic network congestion games; Cost of anarchy

Problem Definition

Consider having a set of resources E in a system. For each $e \in E$, let $d_e(\cdot)$ be the delay per user that requests its service, as a function of the total usage of this resource by all the users. Each such function is considered to be non–decreasing in the total usage of the corresponding resource. Each resource may be represented by a pair of points: an entry point to the resource and an exit point from it. So, each resource is represented by an arc from its entry point to its exit point and the model associates with this arc the cost (e. g., the delay as a function of the load of this resource) that each user has to pay if she is served by this resource. The entry/exit points of the resources need not be unique; they may coincide in order to express the possibility of offering joint service to users, that consists of a sequence of resources. Here, denote by V the set of all entry/exit points of the resources in the system. Any nonempty collection of resources corresponding to a directed path in $G \equiv (V, E)$ comprises an *action* in the system.

Let $N \equiv [n]$ be a set of users, each willing to adopt some action in the system. $\forall i \in N$, let w_i denote user i's *demand* (e. g., the flow rate from a source node to a destination node), while $\Pi_i \subseteq 2^E \setminus \emptyset$ is the collection of actions, any of which would satisfy user i (e. g., alternative routes from a source to a destination node, if G represents a communication network). The collection Π_i is called the *action set* of user i and each of its elements contains at least one resource. Any vector $\boldsymbol{r} = (r_1, \ldots, r_n) \in \Pi \equiv \times_{i=1}^n \Pi_i$ is a *pure strategies profile*, or a *configuration* of the users. Any vector of real functions $\boldsymbol{p} = (p_1, p_2, \ldots, p_n)$ s.t. $\forall i \in [n], p_i : \Pi_i \to [0, 1]$ is a probability distribution over the set of allowable actions for user i (i. e., $\sum_{r_i \in \Pi_i} p_i(r_i) = 1$), and is called a *mixed strategies profile* for the n users.

A congestion model typically deals with users of identical demands, and thus, user cost function depending on the *number* of users adopting each action ([1,4,6]). In this work the more general case is considered, where a *weighted congestion model* is the tuple $((w_i)_{i\in N}, (\Pi_i)_{i\in N}, (d_e)_{e\in E})$. That is, the users are allowed to have different demands for service from the whole system, and thus affect the resource delay functions in a different way, depending on their own weights. A *weighted congestion game* associated with this model, is a game in strategic form with the set of users N and user demands $(w_i)_{i\in N}$, the action sets $(\Pi_i)_{i\in N}$ and cost functions $(\lambda^i_{r_i})_{i\in N, r_i\in\Pi_i}$ defined as follows: For any configuration $\boldsymbol{r} \in \Pi$ and $\forall e \in E$, let $\Lambda_e(\boldsymbol{r}) = \{i \in N : e \in r_i\}$ be the set of users exploiting resource e according to $\boldsymbol{r}$ (called the *view* of resource e wrt configuration $\boldsymbol{r}$). The *cost* $\lambda^i(\boldsymbol{r})$ *of user i for adopting strategy* $r_i \in \Pi_i$ in a given configuration $\boldsymbol{r}$ is equal to the cumulative *delay* $\lambda_{r_i}(\boldsymbol{r})$ along this path:

$$\lambda^i(\boldsymbol{r}) = \lambda_{r_i}(\boldsymbol{r}) = \sum_{e\in r_i} d_e(\theta_e(\boldsymbol{r})) \qquad (1)$$

where, $\forall e \in E, \theta_e(\boldsymbol{r}) \equiv \sum_{i\in\Lambda_e(\boldsymbol{r})} w_i$ is the load on resource e wrt the configuration $\boldsymbol{r}$.

On the other hand, for a mixed strategies profile $\boldsymbol{p}$, the *expected cost of user i for adopting strategy* $r_i \in \Pi_i$ is

$$\lambda^i_{r_i}(\boldsymbol{p}) = \sum_{\boldsymbol{r}^{-i}\in\Pi^{-i}} P(\boldsymbol{p}^{-i}, \boldsymbol{r}^{-i}) \cdot \sum_{e\in r_i} d_e\left(\theta_e(\boldsymbol{r}^{-i} \oplus r_i)\right) \qquad (2)$$

where, $\boldsymbol{r}^{-i}$ is a configuration of all the users except for user i, $\boldsymbol{p}^{-i}$ is the mixed strategies profile of all users except for i, $\boldsymbol{r}^{-i} \oplus r_i$ is the new configuration with user i choosing strategy r_i, and $P(\boldsymbol{p}^{-i}, \boldsymbol{r}^{-i}) \equiv \prod_{j\in N\setminus\{i\}} p_j(r_j)$ is the occurrence probability of $\boldsymbol{r}^{-i}$.

Remark 1 Here notation is abused a little bit and the model considers the user costs $\lambda^i_{r_i}$ as functions whose exact definition depends on the other users' strategies: In the general case of a mixed strategies profile $\boldsymbol{p}$, (2) is valid and expresses the expected cost of user i wrt $\boldsymbol{p}$, conditioned on the event that i chooses path r_i. If the other users adopt a pure strategies profile $\boldsymbol{r}^{-i}$, we get the special form of (1) that expresses the exact cost of user i choosing action r_i.

A congestion game in which all users are indistinguishable (i. e., they have the same user cost functions) and have the same action set, is called *symmetric*. When each user's action set Π_i consists of sets of resources that comprise (simple) paths between a unique origin-destination pair of nodes (s_i, t_i) in a network $G = (V, E)$, the model refers to a *network congestion game*. If additionally all origin-destination pairs of the users coincide with a unique pair

(s, t) one gets a *single commodity network congestion game* and then all users share exactly the same action set. Observe that a single-commodity network congestion game is not necessarily symmetric because the users may have different demands and thus their cost functions will also differ.

Selfish Behavior

Fix an arbitrary (mixed in general) strategies profile $\boldsymbol{p}$ for a congestion game $((w_i)_{i\in N}, (\Pi_i)_{i\in N}, (d_e)_{e\in E})$. We say that $\boldsymbol{p}$ is a *Nash Equilibrium (NE)* if and only if $\forall i \in N, \forall r_i, \pi_i \in \Pi_i, p_i(r_i) > 0 \Rightarrow \lambda^i_{r_i}(\boldsymbol{p}) \le \lambda^i_{\pi_i}(\boldsymbol{p})$. A configuration $\boldsymbol{r} \in \Pi$ is a *Pure Nash Equilibrium (PNE)* if and only if $(\forall i \in N, \forall \pi_i \in \Pi_i, \lambda_{r_i}(\boldsymbol{r}) \le \lambda_{\pi_i}(\boldsymbol{r^{-i}} \oplus \pi_i)$ where, $\boldsymbol{r}^{-i} \oplus \pi_i$ is the same configuration with $\boldsymbol{r}$ except for user i that now chooses action π_i.

Key Results

In this section the article deals with the existence and tractability of PNE in weighted network congestion games. First, it is shown that it is not always the case that a PNE exists, even for a weighted single-commodity network congestion game with only linear and 2-wise linear (e. g., the maximum of two linear functions) resource delays. In contrast, it is well known ([1,6]) that any unweighted (not necessarily single-commodity, or even network) congestion game has a PNE, for any kind of nondecreasing delays. It should be mentioned that the same result has been independently proved also by [3].

Lemma 1 *There exist instances of weighted single–commodity network congestion games with resource delays being either linear or 2-wise linear functions of the loads, for which there is no PNE.*

Theorem 2 *For any weighted multi–commodity network congestion game with linear resource delays, at least one PNE exists and can be computed in pseudo-polynomial time.*

Proof Fix an arbitrary network $G = (V, E)$ with linear resource/edge delays $d_e(x) = a_e x + b_e$, $e \in E$, $a_e, b_e \ge 0$. Let $\boldsymbol{r} \in \Pi$ be an arbitrary configuration for the corresponding weighted multi–commodity congestion game on G. For the configuration r consider the potential $\Phi(\boldsymbol{r}) = C(\boldsymbol{r}) + W(\boldsymbol{r})$, where

$$C(\boldsymbol{r}) = \sum_{e\in E} d_e(\theta_e(\boldsymbol{r}))\theta_e(\boldsymbol{r}) = \sum_{e\in E}[a_e\theta_e^2(\boldsymbol{r}) + b_e\theta_e(\boldsymbol{r})],$$

and

$$W(\boldsymbol{r}) = \sum_{i=1}^{n}\sum_{e\in r_i} d_e(w_i)w_i = \sum_{e\in E}\sum_{i\in\tilde{}_e(\boldsymbol{r})} d_e(w_i)w_i =$$
$$\sum_{e\in E}\sum_{i\in\tilde{}_e(\boldsymbol{r})}(a_e w_i^2 + b_e w_i)$$

one concludes that

$$\Phi(\boldsymbol{r}') - \Phi(\boldsymbol{r}) = 2w_i[\lambda^i(\boldsymbol{r}') - \lambda^i(\boldsymbol{r})]\ ,$$

Note that the potential is a global system function whose changes are proportional to selfish cost improvements of any user. The global minima of the potential then correspond to configurations in which no user can improve her cost acting unilaterally. Therefore, any weighted multi–commodity network congestion game with linear resource delays admits a PNE. □

Applications

In [5] many experiments have been conducted for several classes of pragmatic networks. The experiments show even faster convergence to pure Nash Equilibria.

Open Problems

The Potential function reported here is polynomial on the loads of the users. It is open whether one can find a purely combinatorial potential , which will allow strong polynomial time for finding Pure Nash equilibria.

Cross References

- ► Best Response Algorithms for Selfish Routing
- ► Computing Pure Equilibria in the Game of Parallel Links
- ► General Equilibrium

Recommended Reading

1. Fabrikant A., Papadimitriou C., Talwar K.: The complexity of pure nash equilibria. In: Proc. of the 36th ACM Symp. on Theory of Computing (STOC '04). ACM, Chicago (2004)
2. Fotakis, D., Kontogiannis, S., Spirakis, P.: Selfish unsplittable flows. J. Theoret. Comput. Sci. **348**, 226–239 (2005)
3. Libman L., Orda A.: Atomic resource sharing in noncooperative networks. Telecommun. Syst. **17**(4), 385–409 (2001)
4. Monderer D., Shapley L.: Potential games. Games Eco. Behav. **14**, 124–143 (1996)
5. Panagopoulou P., Spirakis P.: Algorithms for pure Nash Equilibrium in weighted congestion games. ACM J. Exp. Algorithms **11**, 2.7 (2006)
6. Rosenthal R.W.: A class of games possessing pure-strategy nash equilibria. Int. J. Game Theory **2**, 65–67 (1973)

Self-Stabilization

1974; Dijkstra

TED HERMAN
Department of Computer Science, University of Iowa, Iowa City, IA, USA

Keywords and Synonyms

Autopoesis; Homeostasis; Autonomic system control

Problem Definition

An algorithm is self-stabilizing if it eventually manifests correct behavior regardless of initial state. The general problem is to devise self-stabilizing solutions for a specified task. The property of self-stabilization is now known to be feasible for a variety of tasks in distributed computing. Self-stabilization is important for distributed systems and network protocols subject to transient faults. Self-stabilizing systems automatically recover from faults that corrupt state.

The operational interpretation of self-stabilization is depicted in Fig. 1. Part (a) of the figure is an informal presentation of the behavior of a self-stabilizing system, with time on the x-axis and some informal measure of correctness on the y-axis. The curve illustrates a system trajectory, through a sequence of states, during execution. At the initial state, the system state is incorrect; later, the system enters a correct state, then returns to an incorrect state, and subsequently stabilizes to an indefinite period where all states are correct. This period of stability is disrupted by a transient fault that moves the system to an incorrect state, after which the scenario above repeats. Part (b) of the figure illustrates the scenario in terms of state predicates. The box represents the predicate *true*, which characterizes all possible states. Predicate C characterizes the correct states of the system, and $\mathcal{L} \subset C$ depicts the closed *legitimacy* predicate. Reaching a state in $\mathcal{L}$ corresponds to entering a period of stability in part (a). Given an algorithm A with this type of behavior, it is said that A self-stabilizes to $\mathcal{L}$; when $\mathcal{L}$ is implicitly understood, the statement is simplified to: A is self-stabilizing.

Problem [3]. The first setting for self-stabilization posed by Dijkstra is a ring of n processes numbered 0 through $n-1$. Let the state of process i be denoted by $x[i]$. Communication is unidirectional in the ring using a *shared state* model. An atomic step of process i can be expressed by a guarded assignment of the form $g(x[i \ominus 1], x[i]) \rightarrow x[i] := f(x[i \ominus 1], x[i])$. Here, $\ominus$ is subtraction modulo n, so that $x[i \ominus 1]$ is the state of the previous process in the ring with respect to process i. The guard g is a boolean expression; if $g(x[i \ominus 1], x[i])$ is *true*, then process i is said to be *privileged* (or enabled). Thus in one atomic step, privileged process i reads the state of the previous process and computes a new state. Execution scheduling is controlled by a *central daemon*, which fairly chooses one among all enabled processes to take the next step. The problem is to devise g and f so that, regardless of initial states of $x[i]$, $0 \leq i < n$, eventually there is one privilege and every process enjoys a privilege infinitely often.

Complexity Metrics

The complexity of self-stabilization is evaluated by measuring the resource needed for convergence from an arbitrary initial state. Most prominent in the literature of self-stabilization are metrics for worst-case time of convergence and space required by an algorithm solving the given task. Additionally, for reactive self-stabilizing algorithms, metrics are evaluated for the stable behavior of the algorithm, that is, starting from a legitimate state, and compared to non-stabilizing algorithms, to measure costs of self-stabilization.

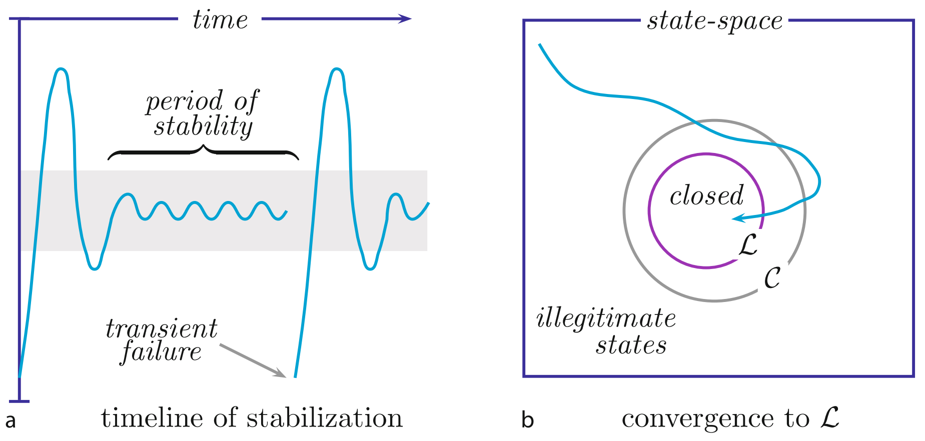
Key Results

Composition

Many self-stabilizing protocols have a layered construction. Let $\{A_i\}_{i=0}^{m-1}$ be a set programs with the property that for every state variable x, if program A_i writes x, then no program A_j, for $j > i$, writes x. Programs in $\{A_j\}_{j=i+1}^{m-1}$ may read variables written by A_i, that is, they use the output of A_i as input. Fair composition of programs B and C, written $B\,[]\,C$, assumes fair scheduling of steps of B and C. Let X_j be the set of variables read by A_j and possibly written by $\{A_i\}_{i=0}^{j-1}$.

Theorem 1 (Fair Composition [4]) *Suppose A_i is self-stabilizing to $\mathcal{L}_i$ under the assumption that all variables in X_i remain constant throughout any execution; then $A_0\,[]\,A_1\,[]\,\cdots\,[]\,A_{m-1}$ self-stabilizes to $\{\mathcal{L}_i\}_{i=0}^{m-1}$.*

Fair composition with a layered set $\{A_i\}_{i=0}^{m-1}$ corresponds to sequential composition of phases in a distributed algorithm. For instance, let B be a self-stabilizing algorithm for mutual exclusion in a network that assumes the existence of a rooted, spanning tree and let algorithm C be a self-stabilizing algorithm to construct a rooted spanning tree in a connected network; then $B\,[]\,C$ is a self-stabilizing mutual exclusion algorithm for a connected network.



Self-Stabilization, Figure 1
Self-stabilization trajectories

Synchronization Tasks

One question related to the problem posed in Sect. "Problem Definition" is whether or not there can be a *uniform* solution, where all processes have identical algorithms. Dijkstra's result for the unidirectional ring is a semi-uniform solution (all but one process have the same algorithm), using n states per process. The state of each process is a counter: process 0 increments the counter modulo k, where $k \geq n$ suffices for convergence; the other processes copy the counter of the preceding process in the ring. At a legitimate state, each time process 0 increments the counter, the resulting value is different from all other counters in the ring. This ring algorithm turns out to be self-stabilizing for the *distributed daemon* (any subset of privileged processes may execute in parallel) when $k > n$. Subsequent results have established that mutual exclusion on a unidirection ring is $\Theta(1)$ space per process with a non-uniform solution. Deterministic uniform solutions to this task are generally impossible, with the exceptional case where n is and prime. Randomized uniform solutions are known for arbitrary n, using $O(\lg \alpha)$ space where α is the smallest number that does not divide n. Some lower bounds on space for uniform solutions are derived in [7]. Time complexity of Dijkstra's algorithm is $O(n^2)$ rounds, and some randomized solutions have been shown to have expected $O(n^2)$ convergence time.

Dijkstra also presented a solution to mutual exclusion for a linear array of processes, using $O(1)$ space per process [3]. This result was later generalized to a rooted tree of processes, but with mutual exclusion relaxed to having one privilege along any path from root to leaf. Subsequent research built on this theme, showing how tasks for distributed wave computations have self-stabilizing solutions. Tasks of phase synchronization and clock synchronization have also been solved. See reference [9] for an example of self-stabilizing mutual exclusion in a multiprocessor shared memory model.

Graph Algorithms

Communication networks are commonly represented with graph models and the need for distributed graph algorithms that tolerate transient faults motivates study of such tasks. Specific results in this area include self-stabilizing algorithms for spanning trees, center-finding, matching, planarity testing, coloring, finding independent sets, and so forth. Generally, all graph tasks can be solved by self-stabilizing algorithms: tasks that have network topology and possibly related factors, such as edge weights, for input, and define outputs to be a function of the inputs, can be solved by general methods for self-stabilization. These general methods require considerable space and time resource, and may also use stronger model assumptions than needed for specific tasks, for instance unique process identifiers and an assumed bound on network diameter. Therefore research continues on graph algorithms.

One discovery emerging from research on self-stabilizing graph algorithms is the difference between algorithms that terminate and those that continuously change state, even after outputs are stable. Consider the task of constructing a spanning tree rooted at process r. Some algorithms self-stabilize to the property that, for every $p \neq r$, the variable u_p refers to p's parent in the spanning tree and the state remains unchanged. Other algorithms are self-stabilizing protocols for token circulation with the side-effect that the circulation route of the token establishes a spanning tree. The former type of algorithm has $O(\lg n)$ space per process, whereas the latter has $O(\lg \delta)$ where δ is the degree (number of neighbors) of a process. This difference was formalized in the notion of *silent* algorithms, which eventually stop changing any communication value; it was shown in [5] for the link register model that silent algorithms for many graph tasks have $\Omega(\lg n)$ space.

Transformation

The simple presentation of [3] is enabled by the abstract computation model, which hides details of communication, program control, and atomicity. Self-stabilization becomes more complicated when considering conventional architectures that have messages, buffers, and program counters. A natural question is how to transform or refine self-stabilizing algorithms expressed in abstract models to concrete models closer to practice. As an example, consider the problem of transforming algorithms written for the central daemon to the distributed daemon model. This transformation can be reduced to finding a self-stabilizing token-passing algorithm for the distributed daemon model such that, eventually, no two neighboring processes concurrently have a token; multiple tokens can increase the efficiency of the transformation.

General Methods

The general problem of constructing a self-stabilizing algorithm for an input nonreactive task can be solved using standard tools of distributed computing: snapshot, broadcast, system reset, and synchronization tasks are building blocks so that the global state can be continuously validated (in some fortunate cases $\mathcal{L}$ can be locally checked and corrected). These building blocks have self-stabilizing solutions, enabling the general approach.

Fault Tolerance

The connection between self-stabilization and transient faults is implicit in the definition. Self-stabilization is also applicable in executions that asynchronously change inputs, silently crash and restart, and perturb communication [10]. One objection to the mechanism of self-stabilization, particularly when general methods are applied, is that a small transient fault can lead to a system-wide correction. This problem has been investigated, for example in [8], where it is shown how convergence can be optimized for a limited number of faults. Self-stabilization has also been combined with other types of failure tolerance, though this is not always possible: the task of counting the number of processes in a ring has no self-stabilizing solution in the shared state model if a process may crash [1], unless a failure detector is provided.

Applications

Many network protocols are self-stabilizing by the following simple strategy: periodically, they discard current data and regenerate it from trusted information sources. This idea does not work in purely asynchronous systems; the availability of real-time clocks enables the simple strategy. Similarly, watchdogs with hardware clocks can provide an effective basis for self-stabilization [6].

Cross References

► Concurrent Programming, Mutual Exclusion

Recommended Reading

1. Anagnostou, E., Hadzilacos, V.: Tolerating Transient and Permanent Failures. In: Distributed Algorithms 7th International Workshop. LNCS, vol. 725, pp. 174–188. Springer, Heidelberg (1993)
2. Cournier, A., Datta, A.K., Petit, F., Villain, V.: Snap-Stabilizing PIF Algorithm in Arbitrary Networks. In: Proceedings of the 22nd International Conference Distributed Computing Systems, pp. 199–206, Vienna, July 2002
3. Dijkstra, E.W.: Self Stabilizing Systems in Spite of Distributed Control. Commun. ACM **17**(11), 643–644 (1974). See also EWD391 (1973) In: Selected Writings on Computing: A Personal Perspective, pp. 41–46. Springer, New York (1982)
4. Dolev, S.: Self-Stabilization. MIT Press, Cambrigde (2000)
5. Dolev, S., Gouda, M.G., Schneider, M.: Memory Requirements for Silent Stabilization. In: Proceedings of the 15th Annual ACM Symposium on Principles of Distributed Computing, pp. 27–34, Philadelphia, May 1996
6. Dolev, S., Yagel, R.: Toward Self-Stabilizing Operating Systems. In: 2nd International Workshop on Self-Adaptive and Autonomic Computing Systems, pp. 684–688, Zaragoza, August 2004
7. Israeli, A., Jalfon, M.: Token Management Schemes and Random Walks Yield Self-Stabilizing Mutual Exclusion. In: Proceedings of the 9th Annual ACM Symposium on Principles of Distributed Computing, pp. 119–131, Quebec City, August 1990
8. Kutten, S., Patt-Shamir, B.: Time-Adaptive Self Stabilization. In: Proceedings of the 16th Annual ACM Symposium on Principles of Distributed Computing, pp. 149–158, Santa Barbara, August 1997
9. Lamport, L.: The Mutual Exclusion Problem: Part II-Statement and Solutions. J. ACM **33**(2), 327–348 (1986)
10. Varghese, G., Jayaram, M.: The Fault Span of Crash Failures. J. ACM **47**(2), 244–293 (2000)

Separators in Graphs

1998; Leighton, Rao
1999; Leighton, Rao

GORAN KONJEVOD
Department of Computer Science and Engineering, Arizona State University, Tempe, AZ, USA

Keywords and Synonyms

Balanced cuts

Problem Definition

The (balanced) separator problem asks for a cut of minimum (edge)-weight in a graph, such that the two shores of the cut have approximately equal (node)-weight.

Formally, given an undirected graph $G = (V, E)$, with a nonnegative edge-weight function $c : E \to \mathbb{R}_+$, a nonnegative node-weight function $\pi : V \to \mathbb{R}_+$, and a constant $b \leq 1/2$, a cut $(S : V \setminus S)$ is said to be *b-balanced*, or a $(b, 1-b)$*-separator*, if $b\pi(V) \leq \pi(S) \leq (1-b)\pi(V)$ (where $\pi(S)$ stands for $\sum_{v \in S} \pi(v)$).

Problem 1 (*b*-balanced separator)
Input*: Edge- and node-weighted graph* $G = (V, E, c, \pi)$, *constant* $b \leq 1/2$.
Output*: A b-balanced cut* $(S : V \setminus S)$. *Goal: minimize the edge weight* $c(\delta(S))$.

Closely related is the *product sparsest cut problem.*

Problem 2 ((Product) Sparsest cut)
Input*: Edge- and node-weighted graph* $G = (V, E, c, \pi)$.
Output*: A cut* $(S : V \setminus S)$ *minimizing the ratio-cost* $\frac{(}{c}(\delta(S)))/(\pi(S)\pi(V \setminus S))$.

Problem 2 is the most general version of sparsest cut solved by Leighton and Rao. Setting all node weights are equal to 1 leads to the uniform version, Problem 3.

Problem 3 ((Uniform) Sparsest cut)
Input*: Edge-weighted graph* $G = (V, E, c)$.
Output*: A cut* $(S : V \setminus S)$ *minimizing the ratio-cost* $(c(\delta(S)))/(|S||V \setminus S|)$.

Sparsest cut arises as the (integral version of the) linear programming dual of *concurrent multicommodity flow* (Problem 4). An instance of a multicommodity flow problem is defined on an edge-weighted graph by specifying for each of k *commodities* a *source* $s_i \in V$, a *sink* $t_i \in V$, and a *demand* D_i. A feasible solution to the multicommodity flow problem defines for each commodity a flow function on E, thus routing a certain amount of flow from s_i to t_i. The edge weights represent capacities, and for each edge e, a capacity constraint is enforced: the sum of all commodities' flows through e is at most the capacity $c(e)$.

Problem 4 (Concurrent multicommodity flow)
Input*: Edge-weighted graph* $G = (V, E, c)$, *commodities* $(s_1, t_1, D_1), \dots (s_k, t_k, D_k)$.
Output*: A multicommodity flow that routes* fD_i *units of commodity i from* s_i *to* t_i *for each i simultaneously, without violating the capacity of any edge. Goal: maximize* f.

Problem 4 can be solved in polynomial time by linear programming, and approximated arbitrarily well by several more efficient combinatorial algorithms (Sect. "Implementation"). The maximum value f for which there exists a multicommodity flow is called the *max-flow* of the instance. The *min-cut* is the minimum ratio $(c(\delta(S)))/(D(S, V \setminus S))$, where $D(S, V \setminus S) = \sum_{i:|\{s_i,t_i\} \cap S|=1} D_i$. This dual interpretation motivates the most general version of the problem, the *nonuniform sparsest cut* (Problem 5).

Problem 5 ((Nonuniform) Sparsest cut) Input*: Edge-weighted graph* $G = (V, E, c)$, *commodities* (s_1, t_1, D_1), $\dots (s_k, t_k, D_k)$.
Output*: A min-cut* $(S : V \setminus S)$*, that is, a cut of minimum ratio-cost* $(c(\delta(S)))/(D(S, V \setminus S))$.

(Most literature focuses on either the uniform or the general nonuniform version, and both of these two versions are sometimes referred to as just the "sparsest cut" problem.)

Key Results

Even when all (edge- and node-) weights are equal to 1, finding a minimum-weight b-balanced cut is NP-hard (for $b = 1/2$, the problem becomes *graph bisection*). Leighton and Rao [23,24] give a pseudo-approximation algorithm for the general problem.

Theorem 1 *There is a polynomial-time algorithm that, given a weighted graph* $G = (V, E, c, \pi)$, $b \leq 1/2$ *and* $b' < \min\{b, 1/3\}$, *finds a* b'*-balanced cut of weight* $O((\log n)/(b - b'))$ *times the weight of the minimum b-balanced cut.*

The algorithm solves the sparsest cut problem on the given graph, puts aside the smaller-weight shore of the cut, and recurses on the larger-weight shore until both shores of the sparsest cut found have weight at most $(1 - b')\pi(G)$. Now the larger-weight shore of the last iteration's sparsest cut is returned as one shore of the balanced cut, and everything else as the other shore. Since the sparsest cut problem is itself NP-hard, Leighton and Rao first required an approximation algorithm for this problem.

Theorem 2 *There is a polynomial-time algorithm with approximation ratio $O(\log p)$ for product sparsest cut (Problem 2), where p denotes the number of nonzero-weight nodes in the graph.*

This algorithm follows immediately from Theorem 3.

Theorem 3 *There is a polynomial-time algorithm that finds a cut $(S : V \setminus S)$ with ratio-cost $(c(\delta(S)))/(\pi(S)\pi(V \setminus S)) \in O(f \log p)$, where f is the max-flow for the product multicommodity flow and p the number of nodes with nonzero weight.*

The proof of Theorem 3 is based on solving a linear programming formulation of the multicommodity flow problem and using the solution to construct a sparse cut.

Related Results

Shahrokhi and Matula [27] gave a max-flow min-cut theorem for a special case of the multicommodity flow problem and used a similar LP-based approach to prove their result. An $O(\log n)$ upper bound for arbitrary demands was proved by Aumann and Rabani [6] and Linial et al. [26]. In both cases, the solution to the dual of the multicommodity flow linear program is interpreted as a finite metric and embedded into ℓ_1 with distortion $O(\log n)$, using an embedding due to Bourgain [10]. The resulting ℓ_1 metric is a convex combination of cut metrics, from which a cut can be extracted with sparsity ratio at least as good as that of the combination.

Arora et al. [5] gave an $O(\sqrt{\log n})$ pseudo-approximation algorithm for (uniform or product-weight) balanced separators, based on a semidefinite programming relaxation. For the nonuniform version, the best bound is $O(\sqrt{\log n}\log\log n)$ due to Arora et al. [4]. Khot and Vishnoi [18] showed that, for the nonuniform version of the problem, the semidefinite relaxation of [5] has an integrality gap of at least $(\log\log n)^{1/6-\delta}$ for any $\delta > 0$, and further, assuming their Unique Games Conjecture, that it is NP-hard to (pseudo)-approximate the balanced separator problem to within any constant factor. The SDP integrality gap was strengthened to $\Omega(\log\log n)$ by Krauthgamer and Rabani [20]. Devanur et al. [11] show an $\Omega(\log\log n)$ integrality gap for the SDP formulation even in the uniform case.

Implementation

The bottleneck in the balanced separator algorithm is solving the multicommodity flow linear program. There exists a substantial amount of work on fast approximate solutions to such linear programs [19,22,25]. In most of the following results, the algorithm produces a $(1+\epsilon)$-approximation, and its hidden constant depends on ϵ^{-2}. Garg and Könemann [15], Fleischer [14] and Karakostas [16] gave efficient approximation schemes for multicommodity flow and related problems, with running times $\tilde{O}((k+m)m)$ [15] and $\tilde{O}(m^2)$ [14,16]. Benczúr and Karger [7] gave an $O(\log n)$ approximation to sparsest cut based on randomized minimum cut and running in time $\tilde{O}(n^2)$. The current fastest $O(\log n)$ sparsest cut (balanced separator) approximation is based on a primal-dual approach to semidefinite programming due to Arora and Kale [3], and runs in time $O(m+n^{3/2})(\tilde{O}(m+n^{3/2})$, respectively). The same paper gives an $O(\sqrt{\log n})$ approximation in time $O(n^2)(\tilde{O}(n^2)$, respectively), improving on a previous $\tilde{O}(n^2)$ algorithm of Arora et al. [2]. If an $O(\log^2 n)$ approximation is sufficient, then sparsest cut can be solved in time $\tilde{O}(n^{3/2})$, and balanced separator in time $\tilde{O}(m+n^{3/2})$ [17].

Applications

Many problems can be solved by using a balanced separator or sparsest cut algorithm as a subroutine. The approximation ratio of the resulting algorithm typically depends directly on the ratio of the underlying subroutine. In most cases, the graph is recursively split into pieces of balanced size. In addition to the $O(\log n)$ approximation factor required by the balanced separator algorithm, this leads to another $O(\log n)$ factor due to the recursion depth. Even et al. [12] improved many results based on balanced separators by using *spreading metrics*, reducing the approximation guarantee to $O(\log n \log\log n)$ from $O(\log^2 n)$.

Some applications are listed here; where no reference is given, and for further examples, see [24].

- Minimum cut linear arrangement and minimum feedback arc set. One single algorithm provides an $O(\log^2 n)$ approximation for both of these problems.
- Minimum chordal graph completion and elimination orderings [1]. Elimination orderings are useful for solving sparse symmetric linear systems. The $O(\log^2 n)$ approximation algorithm of [1] for chordal graph completion has been improved to $O(\log n \log\log n)$ by Even et al. [12].
- Balanced node cuts. The cost of a balanced cut may be measured in terms of the weight of nodes removed from the graph. The balanced separator algorithm can be easily extended to this node-weighted case.
- VLSI layout. Bhatt and Leighton [8] studied several optimization problems in VLSI layout. Recursive par-

titioning by a balanced separator algorithm leads to polylogarithmic approximation algorithms for crossing number, minimum layout area and other problems.

- Treewidth and pathwidth. Bodlaender et al. [9] showed how to approximate treewidth within $O(\log n)$ and pathwidth within $O(\log^2 n)$ by using balanced node separators.
- Bisection. Feige and Krauthgamer [13] gave an $O(\alpha \log n)$ approximation for the minimum bisection, using any α-approximation algorithm for sparsest cut.

Experimental Results

Lang and Rao [21] compared a variant of the sparsest cut algorithm from [24] to methods used in graph decomposition for VLSI design.

Cross References

▶ Fractional Packing and Covering Problems
▶ Minimum Bisection
▶ Sparsest Cut

Recommended Reading

Further details and pointers to additional results may be found in the survey [28].

1. Agrawal, A., Klein, P.N., Ravi, R.: Cutting down on fill using nested dissection: provably good elimination orderings. In: Brualdi, R.A., Friedland, S., Klee, V. (eds.) Graph theory and sparse matrix computation. IMA Volumes in mathematics and its applications, pp. 31–55. Springer, New York (1993)
2. Arora, S., Hazan, E., Kale, S.: $O(\sqrt{\log n})$ approximation to sparsest cut in $\tilde{O}(n^2)$ time. In: FOCS '04: Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS'04), pp. 238–247. IEEE Computer Society, Washington (2004)
3. Arora, S., Kale, S.: A combinatorial, primal-dual approach to semidefinite programs. In: STOC '07: Proceedings of the 39th Annual ACM Symposium on Theory of Computing, pp. 227–236. ACM (2007)
4. Arora, S., Lee, J.R., Naor, A.: Euclidean distortion and the sparsest cut. In: STOC '05: Proceedings of the thirty-seventh annual ACM symposium on Theory of computing, pp. 553–562. ACM Press, New York (2005)
5. Arora, S., Rao, S., Vazirani, U.: Expander flows, geometric embeddings and graph partitioning. In: STOC '04: Proceedings of the thirty-sixth annual ACM symposium on Theory of computing, pp. 222–231. ACM Press, New York (2004)
6. Aumann, Y., Rabani, Y.: An (log) approximate min-cut max-flow theorem and approximation algorithm. SIAM J. Comput. **27**(1), 291–301 (1998)
7. Benczúr, A.A., Karger, D.R.: Approximating s-t minimum cuts in $\tilde{O}(n^2)$ time. In: STOC '96: Proceedings of the twenty-eighth annual ACM symposium on Theory of computing, pp. 47–55. ACM Press, New York (1996)
8. Bhatt, S.N., Leighton, F.T.: A framework for solving vlsi graph layout problems. J. Comput. Syst. Sci. **28**(2), 300–343 (1984)
9. Bodlaender, H.L., Gilbert, J.R., Hafsteinsson, H., Kloks, T.: Approximating treewidth, pathwidth, frontsize, and shortest elimination tree. J. Algorithms **18**(2), 238–255 (1995)
10. Bourgain, J.: On Lipshitz embedding of finite metric spaces in Hilbert space. Israel J. Math. **52**, 46–52 (1985)
11. Devanur, N.R., Khot, S.A., Saket, R., Vishnoi, N.K.: Integrality gaps for sparsest cut and minimum linear arrangement problems. In: STOC '06: Proceedings of the thirty-eighth annual ACM symposium on Theory of computing, pp. 537–546. ACM Press, New York (2006)
12. Even, G., Naor, J.S., Rao, S., Schieber, B.: Divide-and-conquer approximation algorithms via spreading metrics. J. ACM **47**(4), 585–616 (2000)
13. Feige, U., Krauthgamer, R.: A polylogarithmic approximation of the minimum bisection. SIAM J. Comput. **31**(4), 1090–1118 (2002)
14. Fleischer, L.: Approximating fractional multicommodity flow independent of the number of commodities. SIAM J. Discret. Math. **13**(4), 505–520 (2000)
15. Garg, N., Könemann, J.: Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In: FOCS '98: Proceedings of the 39th Annual Symposium on Foundations of Computer Science, p. 300. IEEE Computer Society, Washington (1998)
16. Karakostas, G.: Faster approximation schemes for fractional multicommodity flow problems. In: SODA '02: Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms, pp. 166–173. Society for Industrial and Applied Mathematics, Philadelphia (2002)
17. Khandekar, R., Rao, S., Vazirani, U.: Graph partitioning using single commodity flows. In: STOC '06: Proceedings of the thirty-eighth annual ACM symposium on Theory of computing, pp. 385–390. ACM Press, New York (2006)
18. Khot, S., Vishnoi, N.K.: The unique games conjecture, integrality gap for cut problems and embeddability of negative type metrics into l_1. In: FOCS '07: Proceedings of the 46th Annual IEEE Symposium on Foundations and Computer Science, pp. 53–62. IEEE Computer Society (2005)
19. Klein, P.N., Plotkin, S.A., Stein, C., Tardos, É.: Faster approximation algorithms for the unit capacity concurrent flow problem with applications to routing and finding sparse cuts. SIAM J. Comput. **23**(3), 466–487 (1994)
20. Krauthgamer, R., Rabani, Y.: Improved lower bounds for embeddings into l_1. In: SODA '06: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm, pp. 1010–1017. ACM Press, New York (2006)
21. Lang, K., Rao, S.: Finding near-optimal cuts: an empirical evaluation. In: SODA '93: Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms, pp. 212–221. Society for Industrial and Applied Mathematics, Philadelphia (1993)
22. Leighton, F.T., Makedon, F., Plotkin, S.A., Stein, C., Stein, É., Tragoudas, S.: Fast approximation algorithms for multicommodity flow problems. J. Comput. Syst. Sci. **50**(2), 228–243 (1995)
23. Leighton, T., Rao, S.: An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms. In: Proceedings of the 29th Annual Symposium on Foundations of Computer Science, pp. 422–431, IEEE Computer Society (1988)

24. Leighton, T., Rao, S.: Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. J. ACM **46**(6), 787–832 (1999)
25. Leong, T., Shor, P., Stein, C.: Implementation of a combinatorial multicommodity flow algorithm. In: Johnson, D.S., McGeoch, C.C. (eds.) Network flows and matching. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 12, pp. 387–406. AMS, Providence (1991)
26. Linial, N., London, E., Rabinovich, Y.: The geometry of graphs and some of its algorithmic applications. Comb. **15**(2), 215–245 (1995)
27. Shahrokhi, F., Matula, D.W.: The maximum concurrent flow problem. J. ACM **37**(2), 318–334 (1990)
28. Shmoys, D.B.: Cut problems and their applications to divide-and-conquer. In: Hochbaum, D.S. (ed.) Approximation algorithms for NP-hard problems, pp. 192–235. PWS Publishing Company, Boston, MA (1997)

Sequential Approximate String Matching

2003; Crochemore, Landau, Ziv-Ukelson
2004; Fredriksson, Navarro

Gonzalo Navarro
Department of Computer Science, University of Chile, Santiago, Chile

Keywords and Synonyms

String matching allowing errors or differences; Inexact string matching; Semiglobal or semilocal sequence similarity

Problem Definition

Given a *text string* $T = t_1 t_2 \dots t_n$ and a *pattern string* $P = p_1 p_2 \dots p_m$, both being sequences over an alphabet Σ of size σ, and given a *distance function* among strings d and a *threshold* k, the *approximate string matching (ASM)* problem is to find all the text positions that finish a so-called *approximate occurrence* of P in T, that is, compute the set $\{j, \exists i, 1 \leq i \leq j, d(P, t_i \dots t_j) \leq k\}$. In the sequential version of the problem T, P, and k are given together, whereas the algorithm can be tailored for a specific d.

The solutions to the problem vary widely depending on the distance d used. This entry focuses on a very popular one, called *Levenshtein distance* or *edit distance*, defined as the minimum number of character insertions, deletions, and substitutions necessary to convert one string into the other. It will also pay some attention to other common variants such as *indel distance*, where only insertions and deletions are permitted and is the dual of the *longest common subsequence lcs* ($d(A, B) = |A| + |B| - 2 \cdot lcs(A, B)$); and *Hamming distance*, where only substitutions are permitted.

A popular generalization of all the above is the *weighted edit distance*, where the operations are given positive real-valued weights and the distance is the minimum sum of weights of a sequence of operations converting one string into the other. The weight of deleting a character c is written $w(c \to \epsilon)$, that of inserting c is written $w(\epsilon \to c)$, and that of substituting c by $c' \neq c$ is written $w(c \to c')$. It is assumed $w(c \to c) = 0$ and the triangle inequality, that is, $w(x \to y) + w(y \to z) \geq w(x \to z)$ for any $x, y, z \in \Sigma \cup \{\epsilon\}$. As the distance may now be asymmetric, it is fixed that $d(A, B)$ is the cost of converting A into B. Of course any result for weighted edit distance applies to edit, Hamming and indel distances (collectively termed *unit-cost edit distances*) as well, but other reductions are not immediate.

Both worst- and average-case complexity are considered. For the latter one assumes that pattern and text are randomly generated by choosing each character uniformly and independently from Σ. For simplicity and practicality, $m = o(n)$ is assumed in this entry.

Key Results

The most ancient and versatile solution to the problem [13] builds over the process of computing weighted edit distance. Let $A = a_1 a_2 \dots a_m$ and $B = b_1 b_2 \dots b_n$ be two strings. Let $C[0 \dots m, 0 \dots n]$ be a matrix such that $C[i, j] = d(a_1 \dots a_i, b_1 \dots b_j)$. Then it holds $C[0, 0] = 0$ and

$$\begin{aligned} C[i, j] = \min(&C[i-1, j] + w(a_i \to \epsilon), C[i, j-1] \\ &+ w(\epsilon \to b_j), C[i-1, j-1] + w(a_i \to b_j)) , \end{aligned}$$

where $C[i, -1] = C[-1, j] = \infty$ is assumed. This matrix is computed in $O(mn)$ time and $d(A, B) = C[m, n]$. In order to solve the approximate string matching problem, one takes $A = P$ and $B = T$, and sets $C[0, j] = 0$ for all j, so that the above formula is used only for $i > 0$.

Theorem 1 (Sellers 1980 [13]) *There exists an $O(mn)$ worst-case time solution to the ASM problem under weighted edit distance.*

The space is $O(m)$ if one realizes that C can be computed column-wise and only column $j - 1$ is necessary to compute column j. As explained, this immediately implies that searching under unit-cost edit distances can be done in $O(mn)$ time as well. In those cases, it is quite easy to compute only part of matrix C so as to achieve $O(kn)$ average-time algorithms [14].

Yet, there exist algorithms with lower worst-case complexity for weighted edit distance. By applying a Ziv-Lempel parsing to P and T, it is possible to identify regions of matrix C corresponding to substrings of P and T that can be computed from other previous regions corresponding to similar substrings of P and T [5].

Theorem 2 (Crochemore et al. 2003 [5]) *There exists an $O(n + mn/\log_\sigma n)$ worst-case time solution to the ASM problem under weighted edit distance. Moreover, the time is $O(n + mnh/\log n)$, where $0 \le h \le \log \sigma$ is the entropy of T.*

This result is very general, also holding for computing weighted edit distance and local similarity (see section on applications). For the case of edit distance and exploiting the unit-cost RAM model, it is possible to do better. On one hand, one can apply a four-Russian technique: All the possible blocks (submatrices of C) of size $t \times t$, for $t = O(\log_\sigma n)$, are precomputed and matrix C is computed block-wise [9]. On the other hand, one can represent each cell in matrix C using a constant number of bits (as it can differ from neighboring cells by ± 1) so as to store and process several cells at once in a single machine word [10]. This latter technique is called *bit-parallelism* and assumes a machine word of $\Theta(\log n)$ bits.

Theorem 3 (Masek and Paterson 1980 [9]; Myers 1999 [10]) *There exist $O(n + mn/(\log_\sigma n)^2)$ and $O(n + mn/\log n)$ worst-case time solutions to the ASM problem under edit distance.*

Both complexities are retained for indel distance, yet not for Hamming distance.

For unit-cost edit distances, the complexity can depend on k rather than on m, as $k < m$ for the problem to be nontrivial and usually k is a small fraction of m (or even $k = o(m)$). A classic technique [8] computes matrix C by processing in constant time diagonals $C[i + d, j + d]$, $0 \le d \le s$, along which cell values do not change. This is possible by preprocessing the suffix trees of T and P for Lowest Common Ancestor queries.

Theorem 4 (Landau and Vishkin 1989 [8]) *There exists an $O(kn)$ worst-case time solution to the ASM problem under unit-cost edit distances.*

Other solutions exist which are better for small k, achieving time $O(n(1 + k^4/m))$ [4]. For the case of Hamming distance, one can achieve improved results using convolutions [1].

Theorem 5 (Amir et al. 2004 [1]) *There exist $O(n\sqrt{k \log k})$ and $O(n(1 + k^3/m)\log k)$ worst-case time solution to the ASM problem under Hamming distance.*

The last result for edit distance [4] achieves $O(n)$ time if k is small enough ($k = O(m^{1/4})$). It is also possible to achieve $O(n)$ time on unit-cost edit distances at the expense of an exponential additive term on m or k: The number of different columns in C is independent of n, so the transition from every possible column to the next can be precomputed as a finite-state machine.

Theorem 6 (Ukkonen 1985 [14]) *There exists an $O(n + m \min(3^m, m(2m\sigma)^k))$ worst-case time solution to the ASM problem under edit distance.*

Similar results apply for Hamming and indel distance, where the exponential term reduces slightly according to the particularities of the distances.

The worst-case complexity of the ASM problem is of course $\Omega(n)$, but it is not known if this can be attained for any m and k. Yet, the average-case complexity of the problem is known.

Theorem 7 (Chang and Marr 1994 [3]) *The average-case complexity of the ASM problem is $\Theta(n(k + \log_\sigma m)/m)$ under unit-cost edit distances.*

It is not hard to prove the lower bound as an extension to Yao's bound for exact string matching [15]. The lower bound was reached in the same paper [3], for $k/m < 1/3 - O(1/\sqrt{\sigma})$. This was improved later to $k/m < 1/2 - O(1/\sqrt{\sigma})$ [6] using a slightly different idea. The approach is to precompute the minimum distance to match every possible text substring (block) of length $O(\log_\sigma m)$ inside P. Then, a text window is scanned backwards, block-wise, adding up those minimum precomputed distances. If they exceed k before scanning all the window, then no occurrence of P with k errors can contain the scanned blocks and the window can be safely slid over the scanned blocks, advancing in T. This is an example of a *filtration* algorithm, which discards most text areas and applies an ASM algorithm only over those areas that cannot be discarded.

Theorem 8 (Fredriksson and Navarro 2004 [6]) *There exists an optimal-on-average solution to the ASM problem under edit distance, for any $k/m \le \frac{1-e/\sqrt{\sigma}}{2-e/\sqrt{\sigma}} = 1/2 - O(1/\sqrt{\sigma})$.*

The result applies verbatim to indel distance. The same complexity is achieved for Hamming distance, yet the limit on k/m improves to $1 - 1/\sigma$. Note that, when the limit k/m is reached, the average complexity is already $\Theta(n)$. It is not clear up to which k/m limit could one achieve linear time on average.

Applications

The problem has many applications in computational biology (to compare DNA and protein sequences, recovering from experimental errors, so as to spot mutations or predict similarity of structure or function), text retrieval (to recover from spelling, typing or automatic recognition errors), signal processing (to recover from transmission and distortion errors), and several others. See [11] for a more detailed discussion.

Many extensions of the ASM problem exist, particularly in computational biology. For example, it is possible to substitute whole substrings by others (called *generalized edit distance*), swap characters in the strings (*string matching with swaps or transpositions*), reverse substrings (*reversal distance*), have variable costs for insertions/deletions when they are grouped (*similarity with gap penalties*), and look for any pair of substrings of both strings that are sufficiently similar (*local similarity*). See for example Gusfield's book [7], where many related problems are discussed.

Open Problems

The worst-case complexity of the problem is not fully understood. For unit-cost edit distances it is $\Theta(n)$ if $m = O(\min(\log n, (\log_\sigma n)^2))$ or $k = O(\min(m^{1/4}, \log_{m\sigma} n))$. For weighted edit distance the complexity is $\Theta(n)$ if $m = O(\log_\sigma n)$. It is also unknown up to which k/m value can one achieve $O(n)$ average time; up to now this has been achieved up to $k/m = 1/2 - O(1/\sqrt{\sigma})$.

Experimental Results

A thorough survey on the subject [11] presents extensive experiments. Nowadays, the fastest algorithms for edit distance are in practice filtration algorithms [6,12] combined with bit-parallel algorithms to verify the candidate areas [2,10]. Those filtration algorithms work well for small enough k/m, otherwise the bit-parallel algorithms should be used stand-alone. Filtration algorithms are easily extended to handle multiple patterns searched simultaneously.

URL to Code

Well-known packages offering efficient ASM are *agrep* (http://webglimpse.net/download.html, top-level subdirectory `agrep/`) and *nrgrep* (http://www.dcc.uchile.cl/~gnavarro/software).

Cross References

- ▶ Approximate Regular Expression Matching is the more complex case where P can be a regular expression;
- ▶ Indexed Approximate String Matching refers to the case where the text can be preprocessed;
- ▶ Local Alignment (with Concave Gap Weights) refers to a more complex weighting scheme of interest in computational biology.
- ▶ Sequential Exact String Matching is the simplified version where no errors are permitted;

Recommended Reading

1. Amir, A., Lewenstein, M., Porat, E.: Faster algorithms for string matching with k mismatches. J. Algorithms **50**(2), 257–275 (2004)
2. Baeza-Yates, R., Navarro, G.: Faster approximate string matching. Algorithmica **23**(2), 127–158 (1999)
3. Chang, W., Marr, T.: Approximate string matching and local similarity. In: Proc. 5th Annual Symposium on Combinatorial Pattern Matching (CPM'94). LNCS, vol. 807, pp. 259–273. Springer, Berlin, Germany (1994)
4. Cole, R., Hariharan, R.: Approximate string matching: A simpler faster algorithm. SIAM J. Comput. **31**(6), 1761–1782 (2002)
5. Crochemore, M., Landau, G., Ziv-Ukelson, M.: A subquadratic sequence alignment algorithm for unrestricted scoring matrices. SIAM J. Comput. **32**(6), 1654–1673 (2003)
6. Fredriksson, K., Navarro, G.: Average-optimal single and multiple approximate string matching. ACM J. Exp. Algorithms **9**(1.4) (2004)
7. Gusfield, D.: Algorithms on strings, trees and sequences. Cambridge University Press, Cambridge (1997)
8. Landau, G., Vishkin, U.: Fast parallel and serial approximate string matching. J. Algorithms **10**, 157–169 (1989)
9. Masek, W., Paterson, M.: A faster algorithm for computing string edit distances. J. Comput. Syst. Sci. **20**, 18–31 (1980)
10. Myers, G.: A fast bit-vector algorithm for approximate string matching based on dynamic progamming. J. ACM **46**(3), 395–415 (1999)
11. Navarro, G.: A guided tour to approximate string matching. ACM Comput. Surv. **33**(1), 31–88 (2001)
12. Navarro, G., Baeza-Yates, R.: Very fast and simple approximate string matching. Inf. Proc. Lett. **72**, 65–70 (1999)
13. Sellers, P.: The theory and computation of evolutionary distances: pattern recognition. J. Algorithms **1**, 359–373 (1980)
14. Ukkonen, E.: Finding approximate patterns in strings. J. Algorithms **6**, 132–137 (1985)
15. Yao, A.: The complexity of pattern matching for a random string. SIAM J. Comput. **8**, 368–387 (1979)

Sequential Circuit Technology Mapping

1998; Pan, Liu

PEICHEN PAN
Magma Design Automation, Inc., Los Angeles, CA, USA

Keywords and Synonyms

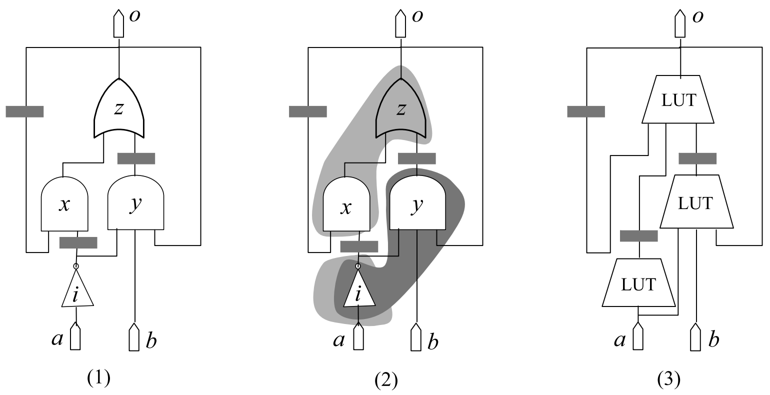
Integrated retiming and technology mapping; Technology mapping with retiming

Problem Definition

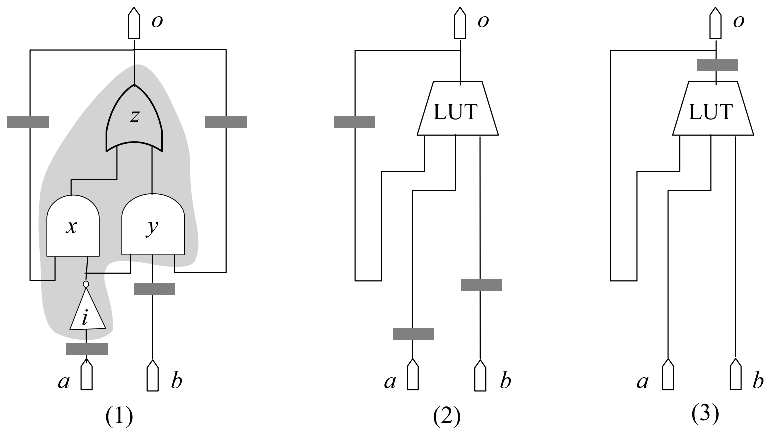
One of the key steps in VLSI design flow is technology mapping which converts a Boolean network of technology-independent logic gates and edge-triggered D-flipflops (FFs) into an equivalent one comprised of cells from a target technology cell library [1,3,5]. Technology mapping can be formulated as a covering problem in where logic gates are covered by cells from the technology library. For ease of discussion, it is assumed that the cell library contains only one cell, a K-input lookup table (K-LUT) with one unit of delay. A K-LUT can realize any Boolean function with up to K inputs as is the case in high performance field-programmable gate arrays (FPGAs).

Figure 1 shows an example of technology mapping. The original network in (1) with three FFs and four gates, is covered by three 3-input cones as indicated in (2). The corresponding mapping solution using 3-LUTs is shown in (3). Note that gate i is covered by two cones. The mapping solution in (3) has a *cycle time* (or *clock period*) of two units, which is the total delay of a longest path between FFs, from primary inputs (PIs) to FFs, and from FFs to primary outputs (POs).

Retiming is a transformation that relocates FFs of a design while preserving its functionality [4]. Retiming can affect technology mapping. Figure 2 (1) shows a design obtained from the one in Fig. 1 (1) by retiming the FFs at the output of y and i to their inputs. It can be covered with just one 3-input cone as indicated in (1). The corresponding mapping solution shown in (2) is better in both timing and area than the functionally-equivalent solution in Fig. 1 (3) obtained without retiming.



Sequential Circuit Technology Mapping, Figure 1
Technology mapping: (1) Original network, (2) covering, (3) mapping solution



Sequential Circuit Technology Mapping, Figure 2
Retiming and mapping: (1) Retiming and covering, (2) mapping solution, (3) retimed solution

```
FindAllCuts($N, K$)
    foreach node $v$ in $N$ do $C(v) \Leftarrow \{\{v^0\}\}$
    while (new cuts discovered) do
        foreach node $v$ in $N$ do $C(v) \Leftarrow merge(C(u_1), ..., C(u_t))$
```

Sequential Circuit Technology Mapping, Figure 3
Cut enumeration procedure

iter	a	b	i	x	y	z	o
0	$\{a^0\}$	$\{b^0\}$	$\{i^0\}$	$\{x^0\}$	$\{y^0\}$	$\{z^0\}$	$\{o^0\}$
1			$\{a^0\}$	$\{i^1, z^1\}$ $\{a^1, z^1\}$	$\{i^0, b^0, z^0\}$ $\{a^0, b^0, z^0\}$	$\{x^0, y^1\}$ $\{i^1, z^1, b^1\}$ $\{a^1, z^1, b^1\}$ $\{i^1, z^1, y^1\}$ $\{a^1, z^1, y^1\}$	$\{z^0\}$
2				$\{i^1, x^1, y^2\}$ $\{a^1, x^1, y^2\}$			

Sequential Circuit Technology Mapping, Figure 4
Cut enumeration example

A *K-bounded* network is one in which each gate has at most K inputs. The sequential circuit technology mapping problem can be defined as follows: *Given a K-bounded Boolean network N and a target cycle time φ , find a mapping solution with a cycle time of φ , assuming FFs can be repositioned using retiming.*

Key Results

The first polynomial time algorithm for the problem was proposed in [8,9]. An improved algorithm was proposed in [2] to reduce runtime. Both algorithms are based on min-cost flow computation.

In [7], another algorithm was proposed to take advantage of the fact that K is a small integer usually between 3 and 6 in practice. The algorithm enumerates all K-input cones for each gate. It can incorporate other optimization objectives (e. g., area and power) and can be apllied to standard cells libraries.

Cut Enumeration

A Boolean network can be represented as an edge-weighted directed graph where the nodes denote logic gates, PIs, and POs. There is a directed edge (u, v) with weight d if u, after going through d FFs, drives v.

A logic cone for a node can be captured by a *cut* consisting of inputs to the cone. An element in a cut for v consists of the driving node u and the total weight d on the paths from u to v, denoted by u^d. If u reaches v on several paths with different FF counts, u will appear in the cut multiple times with different d's. As an example, for the cone for z in Fig. 2 (2), the corresponding cut is $\{z^1, a^1, b^1\}$. A cut of size K is called a K-cut.

Let (u_i, v) be an edge in N with weight d_i, and $C(u_i)$ be a set of K-cuts for u_i, for $i = 1, \dots, t$. Let $merge(C(u_1), \dots, C(u_t))$ denote the following set operation:

$$\{\{v^0\}\} \cup \{c_1^{d_1} \cup \dots \cup c_t^{d_t} | c_1 \in C(u_1), \dots, c_t \in C(u_t),$$
$$|c_1^{d_1} \cup \dots \cup c_t^{d_t}| \le K\}$$

where $c_i^{d_i} = \{u^{d+d_i} | u^d \in c_i\}$ for $i = 1, \dots, t$. It is obvious that $merge(C(u_1), \dots, C(u_t))$ is a set of K-cuts for v.

If the network N does not contain cycles, the K-cuts of all nodes can be determined using the merge operation in

```
FindMinLabels($N$)
    foreach node $v$ in $N$ do $l(v) \Leftarrow -w_v \cdot \phi$
    while(there are updates in labels) do
        foreach node $v$ in $N$ do
            $l(v) \Leftarrow \min_{c \in C(v)}\{\max\{l(u) - d \cdot \phi + 1 | u^d \in c\}\}$
        if $v$ is a PO and $l(v) > \phi$, return failure
    return success
```

Sequential Circuit Technology Mapping, Figure 5
Labeling procedure

iter	*a*	*b*	*i*	*x*	*y*	*z*	*o*
0	$\{a^0\}:0$	$\{b^0\}:0$	$\{i^0\}:0$	$\{x^0\}:-1$	$\{y^0\}:\ 0$	$\{z^0\}:-1$	$\{o^0\}:-1$
1			$\{a^0\}:1$	$\{a^1,z^1\}:0$	$\{a^0,b^0,z^0\}:1$	$\{a^1,z^1,b^1\}:0$	$\{z^0\}:0$

Sequential Circuit Technology Mapping, Figure 6
Labeling example

a topological order starting from the PIs. For general networks, Fig. 3 outlines the iterative cut computation procedure proposed in [7].

Figure 4 depicts the iterations in enumerating 3-cuts for the design in Fig. 1 (1) when cuts are merged in the order i, x, y, z, and o. At the beginning, every node has its trivial cut formed by itself. Row 1 shows the new cuts discovered in the first iteration. In second iteration, two more cuts are discovered (for x). After that, the procedure stops as further merging does not yield any new cut.

Lemma 1 *After at most Kn iterations, the cut enumeration procedure will find the K-cuts for all nodes in N.*

Techniques have been proposed to speed up the procedure [7]. With those techniques, all 4-cuts for each of the ISCAS89 benchmark designs can be found in at most five iterations.

Labeling Phase

After obtaining all K-cuts, the algorithm evaluates the cuts based on sequential arrival times (or l-values), which is an extension of traditional arrival times, to consider the effect of retiming [6,8].

The labeling procedure tries to find a label for each node as outlined in Fig. 5, where w_v denotes the weight of shortest paths from PIs to node v.

Figure 6 shows the iterations for label computation for the design in Fig. 1 (1) assuming the target cycle time $\phi = 1$ and the nodes are evaluated in the order of i, x, y, z, and o. In the table, the current label as well as a corresponding cut for each node is listed. In this example, after first iteration, none of the labels will change and the procedure stops.

It can be shown that the labeling procedure will stop after at most $n(n-1)$ iterations [9]. The following lemma relates labels to mapping:

Lemma 2 *N has a mapping solution with cycle time φ iff the labeling procedure returns "success".*

Mapping Phase

Once the labels for all nodes are computed successfully, a mapping solution can be constructed starting from POs. At each node v, the procedure selects a cut that realizes the label of the node, and then moves on to select a cut for u if u^d is in the cut selected for v. On the edge from the LUT for u to the LUT for v, d FFs are added. For the design in Fig. 1 (1), the mapping solution generated based on the labels found in Fig. 6 is exactly the network in Fig. 2 (2).

To obtain a mapping solution with the target cycle time φ, the LUT for v can be retimed by $\lceil l(v)/\phi \rceil - 1$. For the design in Fig. 1 (1), the final mapping solution after retiming is shown in Fig. 2 (3).

Applications

The algorithm can be used to map a technology-independent Boolean network to a network consisting of cells from a target technology library. The concepts and framework are general enough to be adapted to study other circuit optimizations such as sequential circuit clustering and sequential circuit restructuring.

Cross References

- ▶ Circuit Retiming
- ▶ FPGA Technology Mapping
- ▶ Technology Mapping

Recommended Reading

1. Cong, J., Ding, Y.: FlowMap: An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs. IEEE Trans. on Comput. Aided Des. of Integr. Circuits and Syst., **13**(1), 1–12 (1994)
2. Cong, J., Wu, C.: FPGA Synthesis with Retiming and Pipelining for Clock Period Minimization of Sequential Circuits. ACM/IEEE Design Automation Conference (1997)
3. Keutzer, K.: DAGON: Technology Binding and Local Optimization by DAG Matching. ACM/IEEE Design Automation Conference (1987)
4. Leiserson, C.E., Saxe, J.B.: Retiming Synchronous Circuitry. Algorithmica **6**, 5–35 (1991)
5. Mishchenko, A., Chatterjee, S., Brayton, R., Ciesielski, M.: An integrated technology mapping environment. International Workshop on Logic Synthesis (2005)
6. Pan, P.: Continuous Retiming: Algorithms and Applications. IEEE International Conference on Computer Design, pp. 116–121. (1997)
7. Pan, P., Lin, C.C.: A New Retiming-based Technology Mapping Algorithm for LUT-based FPGAs. ACM International Symposium on Field-Programmable Gate Arrays (1998)

8. Pan, P., Liu, C.L.: Optimal Clock Period FPGA Technology Mapping for Sequential Circuits. ACM/IEEE Design Automation Conference, June (1996)
9. Pan, P., Liu, C.L.: Optimal Clock Period FPGA Technology Mapping for Sequential Circuits. ACM Trans. on Des. Autom. of Electron. Syst., **3**(3), 437–462 (1998)

Sequential Exact String Matching

1994; Crochemore, Czumaj, Gąsieniec, Jarominek, Lecroq, Plandowski, Rytter

Maxime Crochemore[1], Thierry Lecroq[2]
[1] Laboratory of Computer Science, University of Paris-East, Descartes, France
[2] Computer Science Department and LITIS Faculty of Science, University of Rouen, Rouen, France

Keywords and Synonyms

Exact pattern matching

Problem Definition

Given a *pattern string* $P = p_1 p_2 \ldots p_m$ and a *text string* $T = t_1 t_2 \ldots t_n$, both being sequences over an alphabet Σ of size σ, the *exact string matching (ESM)* problem is to find one or, more generally, all the text positions where P occurs in T, that is, compute the set $\{j \mid 1 \leq j \leq n-m+1$ and $P = t_j t_{j+1} \ldots t_{j+m-1}\}$. The pattern is assumed to be given first and is then to be searched for in several texts.

Both worst- and average-case complexity are considered. For the latter one assumes that pattern and text are randomly generated by choosing each character uniformly and independently from Σ. For simplicity and practicality the assumption $m = o(n)$ is set in this entry.

Key Results

Most algorithms that solve the ESM problem proceed in two steps: a preprocessing phase of the pattern P followed by a searching phase over the text T. The preprocessing phase serves to collect information on the pattern in order to speed up the searching phase.

The searching phase of string-matching algorithms works as follows: it first aligns the left ends of the pattern and the text, then compare the aligned symbols of the text and the pattern – this specific work is called an attempt or a scan – and after a whole match of the pattern or after a mismatch it shifts the pattern to the right. It repeats the same procedure again until the right end of the pattern goes beyond the right end of the text. The scanning part can be viewed as operating on the text through a window, which size is most often the length of the pattern. This processing manner is called the scan and shift mechanism. Different scanning strategies of the window lead to algorithms having specific properties and advantages.

The brute force algorithm for the ESM problem consists in checking if P occurs at each position j on T, with $1 \leq j \leq n - m + 1$. It does not need any preprocessing phase. It runs in quadratic time $O(mn)$ with constant extra space and performs $O(n)$ character comparisons on average. This is to be compared with the following bounds.

Theorem 1 (Cole et al. 1995 [3]) *The minimum number of character comparisons to solve the ESM problem in the worst case is* $\geq n + 9/(4m)(n - m)$, *and can be made* $\leq n + 8/(3(m + 1))(n - m)$.

Theorem 2 (Yao 1979 [15]) *The ESM problem can be solved in optimal expected time* $O((\log m/m) \times n)$.

On-Line Text Parsing

The first linear ESM algorithm appears in the 1970's. The preprocessing phase consists in computing the periods of the pattern prefixes, or equivalently the length of the longest border for all the prefixes of the pattern. A border of a string is both a prefix and a suffix of it distinct from the string itself. Let $next[i]$ be the length of the longest border of $p_1 \ldots p_{i-1}$. Consider an attempt at position j, when the pattern $p_1 \ldots p_m$ is aligned with the segment $t_j \ldots t_{j+m-1}$ of the text. Assume that the first mismatch (during a left to right scan) occurs between symbols p_i and t_{i+j} for $1 \leq i \leq m$. Then, $p_1 \ldots p_{i-1} = t_j \ldots t_{i+j-1} = u$ and $a = p_i \neq t_{i+j} = b$. When shifting, it is reasonable to expect that a prefix v of the pattern matches some suffix of the portion u of the text. Doing so, after a shift, the comparisons can resume between $p_{next[i]}$ and t_{i+j} without missing any occurrence of P in T and having to backtrack on the text. There exists two variants, depending on whether $p_{next[i]}$ has to be different from p_i or not.

Theorem 3 (Knuth, Morris and Pratt 1977 [11]) *The text searching can be done in time* $O(n)$ *and space* $O(m)$. *Preprocessing the pattern can be done in time* $O(m)$.

The search can be realized using an implementation with successor by default of the deterministic automaton $\mathcal{D}(P)$ recognizing the language $\Sigma^* P$. The size of the implementation is $O(m)$ independent of the alphabet size, due to the fact that $\mathcal{D}(P)$ possesses $m + 1$ states, m forward arcs, and at most m backward arcs. Using the automaton for searching a text leads to an algorithm having an efficient delay (maximum time for processing a character of the text).

Theorem 4 (Hancart 1993 [10]) *Searching for the pattern P can be done with a delay of* $O(\min\{\sigma, \log_2 m\})$ *letter comparisons.*

Note that for most algorithms the pattern preprocessing is not necessarily done before the text parsing as it can be performed on the fly during the parsing.

Practically-Efficient Algorithms

The Boyer–Moore algorithm is among the most efficient ESM algorithms. A simplified version of it, or the entire algorithm, is often implemented in text editors for the search and substitute commands.

The algorithm scans the characters of the window from right to left beginning with its rightmost symbol. In case of a mismatch (or a complete match of the pattern) it uses two precomputed functions to shift the pattern to the right. These two shift functions are called the *bad-character shift* and the *good-suffix shift*. They are based on the following observations. Assume that a mismatch occurs between character $p_i = a$ of the pattern and character $t_{i+j} = b$ of the text during an attempt at position j. Then, $p_{i+1} \dots p_m = t_{i+j+1} \dots t_{j+m} = u$ and $p_i \neq t_{i+j}$. The good-suffix shift consists in aligning the segment $t_{i+j+1} \dots t_{j+m}$ with its rightmost occurrence in P that is preceded by a character different from p_i. Another variant called the *best-suffix shift* consists in aligning the segment $t_{i+j} \dots t_{j+m}$ with its rightmost occurrence in P. Both variants can be computed in time and space $O(m)$ independent of the alphabet size. If there exists no such segment, the shift consists in aligning the longest suffix v of $t_{i+j+1} \dots t_{j+m}$ with a matching prefix of x. The bad-character shift consists in aligning the text character t_{i+j} with its rightmost occurrence in $p_1 \dots p_{m-1}$. If t_{i+j} does not appear in the pattern, no occurrence of P in T can overlap the symbol t_{i+j}, then the left end of the pattern is aligned with the character at position $i + j + 1$. The search can then be done in $O(n/m)$ in the best case.

Theorem 5 (Cole 1994 (see [5,14])) *During the search for a non-periodic pattern P of length m (such that the length of the longest border of P is less than m/2) in a text T of length n, the Boyer-Moore algorithm performs at most 3n comparisons between letters of P and of T.*

Yao's bound can be reached using an indexing structure for the reverse pattern. This is done by the Reverse Factor algorithm also called BDM (for Backward Dawg Matching).

Theorem 6 (Crochemore et al. 1994 [4]) *The search can be done in optimal expected time* $O((\log m/m) \times n)$ *using the suffix automaton or the suffix tree of the reverse pattern.*

A factor oracle can be used instead of an index structure, this is made possible since the only string of length m accepted by the factor oracle of a string w of length m is w itself. This is done by the Backward Oracle Matching (BOM) algorithm of Allauzen, Crochemore and Raffinot [1]. Its behavior in practice is similar to the one of the BDM algorithm.

Time-Space Optimal Algorithms

Algorithms of this type run in linear time (for both preprocessing and searching) and need only constant space in addition to the inputs.

Theorem 7 (Galil and Seiferas 1983 [8]) *The search can be done optimally in time O(n) and constant extra space.*

After Galil and Seiferas' first solution, other solutions are by Crochemore-Perrin [6] and by Rytter [13]. Algorithms rely on a partition of the pattern in two parts; they first search for the right part of the pattern from left to right, and then, if no mismatch occurs, they search for the left part. The partition can be: the perfect factorization [8], the critical factorization [6], or based on the lexicographically maximum suffix of the pattern [13]. Another solution by Crochemore (see [2]) is a variant of KMP [11]: it computes lower bounds of pattern prefixes periods on the fly and requires no preprocessing.

Bit-Parallel Solution

It is possible to use the bit-parallelism technique for ESM.

Theorem 8 (Baeza-Yates & Gonnet 1992; Wu & Manber 1992 (see [5,14])) *If the length m of the string P is smaller than the number of bits of a machine word, the preprocessing phase can be done in time and space* $\Theta(\sigma)$*. The searching phase executes in time* $\Theta(n)$*.*

It is even possible to use this bit-parallelism technique to simulate the BDM algorithm. This is realized by the BNDM (Backward Non-deterministic Dawg Matching) algorithm (see [2,12]).

In practice, when scanning the window from right to left during an attempt, it is sometimes more efficient to only use the bad-character shift. This was first done by the Horspool algorithm (see [2,12]). Other practical efficient algorithms are the Quick Search by Sunday (see [2,12]) and the Tuned Boyer-Moore by Hume and Sunday (see [2,12]).

There exists another method that uses the bit-parallelism technique that is optimal on the average though it consists actually of a filtration method. It considers sparse q-grams and thus avoids to scan a lot of text positions. It is due to Fredriksson and Grabowski [7].

Applications

The methods which are described here apply to the treatment of the natural language, the treatment and analysis of genetic sequences and of musical sequences, the problems of safety related to data flows like virus detection, and the management of textual data bases, to quote only some immediate applications.

Open Problems

There remain only a few open problems on this question. It is still unknown if it is possible to design an average optimal time constant space string matching algorithm. The exact size of the Boyer-Moore automaton is still unknown (see [5]).

Experimental Results

The book of G. Navarro and M. Raffinot [12] is a good introduction and presents an experimental map of ESM algorithms for different alphabet sizes and pattern lengths. Basically, the Shift-Or algorithm is efficient for small alphabets and short patterns, the BNDM algorithm is efficient for medium size alphabets and medium length patterns, the Horspool algorithm is efficient for large alphabets, and the BOM algorithm is efficient for long patterns.

URL to Code

The site monge.univ-mlv.fr/~lecroq/string presents a large number of ESM algorithms (see also [2]). Each algorithm is implemented in C code and a Java applet is given.

Cross References

- ▶ Indexed approximate string matching refers to the case where the text is preprocessed;
- ▶ Regular expression matching is the more complex case where P can be a regular expression.
- ▶ Sequential approximate string matching is the version where errors are permitted;
- ▶ Sequential multiple string matching is the version where a finite set of patterns is searched in a text;

Recommended Reading

1. Allauzen, C., Crochemore, M., Raffinot, M.: Factor oracle: a new structure for pattern matching. In: SOFSEM'99. LNCS, vol. 1725, pp. 291–306. Springer, Berlin (1999)
2. Charras, C., Lecroq, T.: Handbook of exact string matching algorithms. King's College London Publications, London (2004)
3. Cole, R., Hariharan, R., Paterson, M., Zwick, U.: Tighter lower bounds on the exact complexity of string matching. SIAM J. Comput. **24**(1), 30–45 (1995)
4. Crochemore, M., Czumaj, A., Gąsieniec, L., Jarominek, S., Lecroq, T., Plandowski, W., Rytter, W.: Speeding up two string matching algorithms. Algorithmica **12**(4/5), 247–267 (1994)
5. Crochemore, M., Hancart, C., Lecroq, T.: Algorithms on strings. Cambridge University Press, New York (2007)
6. Crochemore, M., Perrin, D.: Two-way string matching. J. ACM **38**(3), 651–675 (1991)
7. Fredriksson, K., Grabowski, S.: Practical and optimal string matching. In: Proceedings of SPIRE'2005. LNCS, vol. 3772, pp. 374–385. Springer, Berlin (2005)
8. Galil, Z., Seiferas, J.: Time-space optimal string matching. J. Comput. Syst. Sci. **26**(3), 280–294 (1983)
9. Gusfield, D.: Algorithms on strings, trees and sequences. Cambridge University Press, Cambridge, UK (1997)
10. Hancart, C.: On Simon's string searching algorithm. Inf. Process. Lett. **47**(2), 95–99 (1993)
11. Knuth, D.E., Morris, J.H. Jr., Pratt, V.R.: Fast pattern matching in strings. SIAM J. Comput. **6**(1), 323–350 (1977)
12. Navarro, G., Raffinot, M.: Flexible Pattern Matching in Strings – Practical on-line search algorithms for texts and biological sequences. Cambridge University Press, Cambridge, Uk (2002)
13. Rytter, W.: On maximal suffixes and constant-space linear-time versions of KMP algorithm. Theor. Comput. Sci. **299**(1–3), 763–774 (2003)
14. Smyth, W.F.: Computing Patterns in Strings. Addison Wesley Longman, Harlow, UK (2002)
15. Yao, A.: The complexity of pattern matching for a random string. SIAM J. Comput. **8**, 368–387 (1979)

Sequential Multiple String Matching

1999; Crochemore, Czumaj, Gąsieniec, Lecroq, Plandowski, Rytter

MAXIME CROCHEMORE[1,2], THIERRY LECROQ[3]
[1] Department of Computer Science, Kings College London, London, UK
[2] Laboratory of Computer Science, University of Paris-East, Paris, France
[3] Computer Science Department and LITIS Faculty of Science, University of Rouen, Rouen, France

Keywords and Synonyms

Dictionary matching

Problem Definition

Given a finite set of k *pattern strings* $\mathcal{P} = \{P^1, P^2, \dots, P^k\}$ and a *text string* $T = t_1 t_2 \dots t_n$, T and the P^is being sequences over an alphabet Σ of size σ, the *multiple string matching (MSM)* problem is to find one or, more generally, all the text positions where a P^i occurs in T. More precisely the problem is to compute the set $\{j \mid \exists i, P^i = t_j t_{j+1} \ldots t_{j+|P^i|-1}\}$, or equivalently the set $\{j \mid \exists i, P^i = t_{j-|P^i|+1} t_{j-|P^i|+2} \ldots t_j\}$. Note that reporting all the occurrences of the patterns may lead to a quadratic output (for example, when P^is and T are drawn from a one-letter alphabet). The length of the shortest pattern in $\mathcal{P}$ is denoted by ℓmin. The patterns are assumed to be given first and are then to be searched for in several texts. This problem is an extension of the exact string matching problem.

Both worst- and average-case complexities are considered. For the latter one assumes that pattern and text are randomly generated by choosing each character uniformly and independently from Σ. For simplicity and practicality the assumption $|P^i| = o(n)$ is set, for $1 \leq i \leq k$, in this entry.

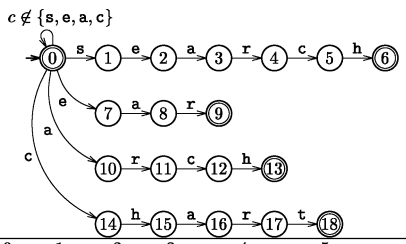
Key Results

A first solution to the multiple string matching problem consists in applying an exact string matching algorithm for locating each pattern in $\mathcal{P}$. This solution has an $O(kn)$ worst case time complexity. There are more efficient solutions along two main approaches. The first one, due to Aho and Corasick [1], is an extension of the automaton-based solution for matching a single string. The second approach, initiated by Commentz-Walter [3], extends the Boyer–Moore algorithm to several patterns.

The Aho–Corasick algorithm first builds a trie $T(\mathcal{P})$, a digital tree recognizing the patterns of $\mathcal{P}$. The trie $T(\mathcal{P})$ is a tree whose edges are labeled by letters and whose branches spell the patterns of $\mathcal{P}$. A node p in the trie $T(\mathcal{P})$ is associated with the unique word w spelled by the path of $T(\mathcal{P})$ from its root to p. The root itself is identified with the empty word ε. Notice that if w is a node in $T(\mathcal{P})$ then w is a prefix of some $P^i \in \mathcal{P}$. If in addition $a \in \Sigma$ then $child(w, a)$ is equal to wa if wa is a node in $T(\mathcal{P})$; it is equal to NIL otherwise.

During a second phase, when patterns are added to the trie, the algorithm initializes an output function *out*. It associates the singleton $\{P^i\}$ with the nodes P^i $(1 \leq i \leq k)$, and associates the empty set with all other nodes of $T(\mathcal{P})$.

Finally, the last phase of the preprocessing consists in building a failure link for each node of the trie, and simultaneously completing the output function. The failure function *fail* is defined on nodes as follows (w is a node): $fail(w) = u$ where u is the longest proper suffix of w that belongs to $T(\mathcal{P})$. Computation of failure links is done during a breadth-first traversal of $T(\mathcal{P})$. Completion of the output function is done while computing the failure function *fail* using the following rule: if $fail(w) = u$ then $out(w) = out(w) \cup out(u)$.



state	0	1	2	3	4	5	6
prefix	ε	s	se	sea	sear	searc	search
fail	0	0	7	8	9	12	13
out					{ear}		{search, arch}
state	7	8	9	10	11	12	13
prefix	e	ea	ear	a	ar	arc	arch
fail	0	10	11	0	0	14	15
out		{ear}				{arch}	
state	14	15	16	17	18	19	
prefix	c	ch	cha	char	chart		
fail	0	0	10	11	0		
out					{chart}		

Sequential Multiple String Matching, Figure 1
The Pattern Matching Machine or Aho–Corasick automaton for the set of strings {search, ear, arch, chart}

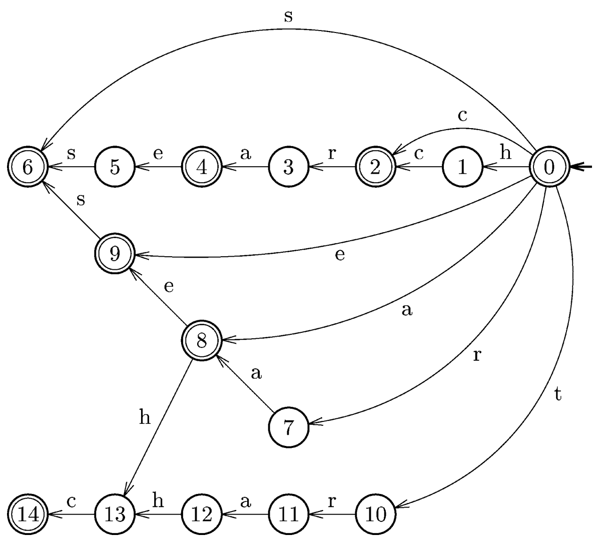
To stop going back with failure links during the computation of the failure links, and also to overpass text characters for which no transition is defined from the root during the searching phase, a loop is added on the root of the trie for these symbols. This finally produces what is called a Pattern Matching Machine or an Aho–Corasick automaton (see Fig. 1).

After the preprocessing phase is completed, the searching phase consists in parsing the text T with $T(\mathcal{P})$. This starts at the root of $T(\mathcal{P})$ and uses failure links whenever a character in T does not match any label of outgoing edges of the current node. Each time a node with a nonempty output is encountered, this means that the patterns of the output have been discovered in the text, ending at the current position. Then, the position is output.

Theorem 1 (Aho and Corasick [1]) *After preprocessing $\mathcal{P}$, searching for the occurrences of the strings of $\mathcal{P}$ in a text T can be done in time $O(n \times \log \sigma)$. The running time of the associated preprocessing phase is $O(|\mathcal{P}| \times \log \sigma)$. The extra memory space required for both operations is $O(|\mathcal{P}|)$.*

The Aho–Corasick algorithm is actually a generalization to a finite set of strings of the Morris–Pratt exact string matching algorithm.

Commentz-Walter [3] generalized the Boyer–Moore exact string matching algorithm to Multiple String Matching. Her algorithm builds a trie for the reverse patterns in $\mathcal{P}$ together with two shift tables, and applies a right to left scan strategy. However it is intricate to implement and has a quadratic worst-case time complexity.



Sequential Multiple String Matching, Figure 2
An example of DAWG, index structure used for matching the set of strings {`search`, `ear`, `arch`, `chart`}. The automaton accepts the reverse prefixes of the strings

The DAWG-match algorithm [4] is a generalization of the BDM exact string matching algorithm. It consists in building an exact indexing structure for the reverse strings of $\mathcal{P}$ such as a factor automaton or a generalized suffix tree, instead as just a trie as in the previous solution (see Fig. 2). The overall algorithm can be made optimal by using both an indexing structure for the reverse patterns and an Aho–Corasick automaton for the patterns. Then, searching involves scanning some portions of the text from left to right and some other portions from right to left. This enables to skip large portions of the text T.

Theorem 2 (Crochemore et al. [4]) *The DAWG-match algorithm performs at most 2n symbol comparisons. Assuming that the sum of the length of the patterns in $\mathcal{P}$ is less than ℓmin^k, the DAWG-match algorithm makes on average $O((n \log \ell min)/\ell min)$ inspections of text characters.*

The bottleneck of the DAWG-match algorithm is the construction time and space consumption of the exact indexing structure. This can be avoided by replacing the exact indexing structure by a factor oracle for a set of strings. When the factor oracle is used alone, it gives the Set Backward Oracle Matching (SBOM) algorithm [2]. It is an exact algorithm that behaves almost as well as the DAWG-match algorithm.

The bit-parallelism technique can be used to simulate the DAWG-match algorithm. It gives the MultiBNDM algorithm of Navarro and Raffinot [7]. This strategy is efficient when $k \times \ell min$ bits fit in a few computer words. The prefixes of strings of $\mathcal{P}$ of length ℓmin are packed together in a bit vector. Then, the search is similar to the BNDM exact string matching and is performed for all the prefixes at the same time.

The use of the generalization of the bad-character shift alone as done in the Horspool exact string matching algorithm gives poor performances for the MSM problem due to the high probability of finding each character of the alphabet in one of the strings of $\mathcal{P}$.

The algorithm of Wu and Manber [11] considers blocks of length ℓ. Blocks of such a length are hashed using a function h into values less than *maxvalue*. Then $shift[h(B)]$ is defined as the minimum between $|P^i| - j$ and $\ell min - \ell + 1$ with $B = p^i_{j-\ell+1} \dots p^i_j$ for $1 \le i \le k$ and $1 \le j \le |P^i|$. The value of ℓ varies with the minimum length of the strings in $\mathcal{P}$ and the size of the alphabet. The value of *maxvalue* varies with the memory space available.

The searching phase of the algorithm consists in reading blocks B of length ℓ. If $shift[h(B)] > 0$ then a shift of length $shift[h(B)]$ is applied. Otherwise, when $shift[h(B)] = 0$ the patterns ending with block B are examined one by one in the text. The first block to be scanned is $t_{\ell min-\ell+1} \dots t_{\ell min}$. This method is incorporated in the *agrep* command [10].

Applications

MSM algorithms serve as basis for multidimensional pattern matching and approximate pattern matching with wildcards. The problem has many applications in computational biology, database search, bibliographic search, virus detection in data flows, and several others.

Experimental Results

The book of G. Navarro and M. Raffinot [8] is a good introduction to the domain. It presents experimental graphics that report experimental evaluation of multiple string matching algorithms for different alphabet sizes, pattern lengths, and sizes of pattern set.

URL to Code

Well-known packages offering efficient MSM are *agrep* (http://webglimpse.net/download.html, top-level subdirectory `agrep/`) and *grep* with the -F option (http://www.gnu.org/software/grep/grep.html).

Cross References

- ► Indexed String Matching refers to the case where the text can be preprocessed;

- ▶ Multidimensional String Matching is the case where the text dimension is greater than one.
- ▶ Regular Expression Matching is the more complex case where the pattern can be a regular expression;
- ▶ Sequential Exact String Matching is the version where a single pattern is searched for in a text;

Recommended Reading

Further information can be found in the four following books: [5,6,8] and [9].

1. Aho, A.V., Corasick, M.J.: Efficient string matching: an aid to bibliographic search. C. ACM **18**(6), 333–340 (1975)
2. Allauzen, C., Crochemore, M., Raffinot, M.: Factor oracle: a new structure for pattern matching. In: SOFSEM'99. LNCS, vol. 1725, pp. 291–306. Springer, Berlin (1999)
3. Commentz-Walter, B.: A string matching algorithm fast on the average. In: Proceedings of ICALP'79. LNCS, vol. 71, pp. 118–132. Springer, Berlin (1979)
4. Crochemore, M., Czumaj, A., Gąsieniec, L., Lecroq, T., Plandowski, W., Rytter, W.: Fast practical multi-pattern matching. Inf. Process. Lett. **71**(3–4), 107–113 (1999)
5. Crochemore, M., Hancart, C., Lecroq, T.: Algorithms on strings. Cambridge University Press, Cambridge (2007)
6. Gusfield, D.: Algorithms on strings, trees and sequences. Cambridge University Press, Cambridge (1997)
7. Navarro, G., Raffinot, M.: Fast and flexible string matching by combining bit-parallelism and suffix automata. ACM J. Exp. Algorithm **5**, 4 (2000)
8. Navarro, G., Raffinot, M.: Flexible Pattern Matching in Strings – Practical on-line search algorithms for texts and biological sequences. Cambridge University Press, Cambridge (2002)
9. Smyth, W.F.: Computing Patterns in Strings. Addison Wesley Longman (2002)
10. Wu, S., Manber, U.: Agrep – a fast approximate pattern-matching tool. In: Proceedings of USENIX Winter (1992) Technical Conference, pp. 153–162. USENIX Association, Berkeley (1992)
11. Wu, S., Manber, U.: A fast algorithm for multi-pattern searching. Report TR-94-17, Department of Computer Science, University of Arizona, Tucson, AZ (1994)

Set Agreement

1993; Chaudhuri

Michel Raynal
IRISA, University of Rennes 1,
Rennes, France

Keywords and Synonyms

Distributed coordination

Problem Definition

Short History

The k-set agreement problem is a paradigm of coordination problems. Defined in the setting of systems made up of processes prone to failures, it is a simple generalization of the consensus problem (that corresponds to the case $k = 1$). That problem was introduced in 1993 by Chaudhuri [2] to investigate how the number of choices (k) allowed for the processes is related to the maximum number of processes that can crash. (After it has crashed, a process executes no more steps: a crash is a premature halting.)

Definition

Let S be a system made up of n processes where up to t can crash and where each process has an input value (called a *proposed* value). The problem is defined by the three following properties (i. e., any algorithm that solves that problem has to satisfy these properties):

1. Termination. Every nonfaulty process decides a value.
2. Validity. A decided value is a proposed value.
3. Agreement. At most k different values are decided.

The Trivial Case

It is easy to see that this problem can be trivially solved if the upper bound on the number of process failures t is smaller than the allowed number of choices k, also called the *coordination degree*. (The trivial solution consists in having $t + 1$ predetermined processes that send their proposed values to all the processes, and a process deciding the first value it ever receives.) So, $k \leq t$ is implicitly assumed in the following.

Key Results

Key Results in Synchronous Systems

The Synchronous Model In this computation model, each execution consists of a sequence of rounds. These are identified by the successive integers 1, 2, etc. For the processes, the current round number appears as a global variable whose global progress entails their own local progress.

During a round, a process first broadcasts a message, then receives messages, and finally executes local computation. The fundamental synchrony property the a synchronous system provides the processes with is the following: a message sent during a round r is received by its destination process during the very same round r. If during a round, a process crashes while sending a message, an arbitrary subset (not known in advance) of the processes receive that message.

```
Function k-set_agreement ($v_i$)
(1)  $est_i \leftarrow v_i$;
(2)  when $r = 1, 2, \ldots, \lfloor \frac{t}{k} \rfloor + 1$ do % $r$: round number %
(3)  begin_round
(4)      send ($est_i$) to all; % including $p_i$ itself %
(5)      $est_i \leftarrow \min(\{est_j$ values received during
                                   the current round $r\})$;
(6)  end_round;
(7)  return ($est_i$)
```

Set Agreement, Figure 1
A simple *k*-set agreement synchronous algorithm (code for p_i)

Main Results The k-set agreement problem can always be solved in a synchronous system. The main result is for the minimal number of rounds (R_t) that are needed for the nonfaulty processes to decide in the worst-case scenario (this scenario is when exactly k processes crash in each round). It was shown in [3] that $R_t = \lfloor \frac{t}{k} \rfloor + 1$. A very simple algorithm that meets this lower bound is described in Fig. 1.

Although failures do occur, they are rare in practice. Let f denote the number of processes that crash in a given run, $0 \leq f \leq t$. We are interested in synchronous algorithms that terminate in at most R_t rounds when t processes crash in the current run, but that allow the nonfaulty processes to decide in far fewer rounds when there are few failures. Such algorithms are called *early-deciding* algorithms. It was shown in [4] that, in the presence of f process crashes, any early-deciding k-set agreement algorithm has runs in which no process decides before the round $R_f = \min(\lfloor \frac{f}{k} \rfloor + 2, \lfloor \frac{t}{k} \rfloor + 1)$. This lower bound shows an inherent tradeoff linking the coordination degree k, the maximum number of process failures t, the actual number of process failures f, and the best time complexity that can be achieved. Early-deciding k-set agreement algorithms for the synchronous model can be found in [4,12].

Other Failure Models In the send omission failure model, a process is faulty if it crashes or forgets to send messages. In the general omission failure model, a process is faulty if it crashes, forgets to send messages, or forgets to receive messages. (A send omission failure models the failure of an output buffer, while a receive omission failure models the failure of an input buffer.) These failure models were introduced in [11].

The notion of *strong* termination for set agreement problems was introduced in [13]. Intuitively, that property requires that as many processes as possible decide. Let a *good* process be a process that neither crashes nor commits receive omission failures. A set agreement algorithm is strongly terminating if it forces all the good processes to decide. (Only the processes that crash during the execution of the algorithm, or that do not receive enough messages, can be prevented from deciding.)

An early-deciding k-set agreement algorithm for the general omission failure model was described in [13]. That algorithm, which requires $t < n/2$, directs a good process to decide and stop in at most $R_f = \min(\lfloor \frac{f}{k} \rfloor + 2, \lfloor \frac{t}{k} \rfloor + 1)$ rounds. Moreover, a process that is not a good process executes at most $R_f(not\_good) = \min(\lceil \frac{f}{k} \rceil + 2, \lfloor \frac{t}{k} \rfloor + 1)$ rounds.

As R_f is a lower bound for the number of rounds in the crash failure model, the previous algorithm shows that R_f is also a lower bound for the nonfaulty processes to decide in the more severe general omission failure model. Proving that $R_f(not\_good)$ is an upper bound for the number of rounds that a nongood process has to execute remains an open problem.

It was shown in [13] that, for a given coordination degree k, $t < \frac{k}{k+1} n$ is an upper bound on the number of process failures when one wants to solve the k-set agreement problem in a synchronous system prone to process general omission failures. A k-set agreement algorithm that meets this bound was described in [13]. That algorithm requires the processes execute $R = t + 2 - k$ rounds to decide. Proving (or disproving) that R is a lower bound when $t < \frac{k}{k+1} n$ is an open problem. Designing an early-deciding k-set agreement algorithm for $t < \frac{k}{k+1} n$ and $k > 1$ is another problem that remains open.

Key Results in Asynchronous Systems

Impossibility A fundamental result of distributed computing is the impossibility to design a deterministic algorithm that solves the k-set agreement problem in asynchronous systems when $k \leq t$ [1,7,15]. Compared with the impossibility of solving asynchronous consensus despite one process crash, that impossibility is based on deep combinatorial arguments. This impossibility has opened new research directions for the connection between distributed computing and topology. This topology approach has allowed the discovery of links relating asynchronous k-set agreement with other distributed computing problems such as the *renaming* problem [5].

Circumventing the Impossibility Several approaches have been investigated to circumvent the previous impossibility. These approaches are the same as those that have been used to circumvent the impossibility of asynchronous consensus despite process crashes.

One approach consists in replacing the "deterministic algorithm" by a "randomized algorithm." In that case, the termination property becomes "the probability for a correct process to decide tends to 1 when the number of rounds tends to $+\infty$." That approach was investigated in [9].

Another approach that has been proposed is based on failure detectors. Roughly speaking, a failure detector provides each process with a list of processes suspected to have crashed. As an example, the class of failure detectors denoted $\Diamond S_x$ includes all the failure detectors such that, after some finite (but unknown) time, (1) any list contains the crashed processes and (2) there is a set Q of x processes such that Q contains one correct process and that correct process is no longer suspected by the processes of Q (let us observe that correct processes can be suspected intermittently or even forever). Tight bounds for the k-set agreement problem in asynchronous systems equipped with such failure detectors, conjectured in [9], were proved in [6]. More precisely, such a failure detector class allows the k-set agreement problem to be solved for $k \geq t - x + 2$ [9], and cannot solve it when $k < t - x + 2$ [6].

Another approach that has been investigated is the combination of failure detectors and conditions [8]. A condition is a set of input vectors, and each input vector has one entry per process. The entries of the input vector associated with a run contain the values proposed by the processes in that run. Basically, such an approach guarantees that the nonfaulty processes always decide when the actual input vector belongs to the condition the k-set algorithm has been instantiated with.

Applications

The set agreement problem was introduced to study how the number of failures and the synchronization degree are related in an asynchronous system; hence, it is mainly a theoretical problem. That problem is used as a canonical problem when one is interested in asynchronous computability in the presence of failures. Nevertheless, one can imagine practical problems the solutions of which are based on the set agreement problem (e. g., allocating a small shareable resources—such as broadcast frequencies—in a network).

Cross References

- ► Asynchronous Consensus Impossibility
- ► Failure Detectors
- ► Renaming
- ► Topology Approach in Distributed Computing

Recommended Reading

1. Borowsky, E., Gafni, E.: Generalized FLP Impossibility Results for *t*-Resilient Asynchronous Computations. In: Proc. 25th ACM Symposium on Theory of Computation, California, 1993, pp. 91–100
2. Chaudhuri, S.: More Choices Allow More Faults: Set Consensus Problems in Totally Asynchronous Systems. Inf. Comput. **105**, 132–158 (1993)
3. Chaudhuri, S., Herlihy, M., Lynch, N., Tuttle, M.: Tight Bounds for *k*-Set Agreement. J. ACM **47**(5), 912–943 (2000)
4. Gafni, E., Guerraoui, R., Pochon, B.: From a Static Impossibility to an Adaptive Lower Bound: The Complexity of Early Deciding Set Agreement. In: Proc. 37th ACM Symposium on Theory of Computing (STOC 2005), pp. 714–722. ACM Press, New York (2005)
5. Gafni, E., Rajsbaum, S., Herlihy, M.: Subconsensus Tasks: Renaming is Weaker than Set Agreement. In: Proc. 20th Int'l Symposium on Distributed Computing (DISC'06). LNCS, vol. 4167, pp. 329–338. Springer, Berlin (2006)
6. Herlihy, M.P., Penso, L.D.: Tight Bounds for *k*-Set Agreement with Limited Scope Accuracy Failure Detectors. Distrib. Comput. **18**(2), 157–166 (2005)
7. Herlihy, M.P., Shavit, N.: The Topological Structure of Asynchronous Computability. J. ACM **46**(6), 858–923 (1999)
8. Mostefaoui, A., Rajsbaum, S., Raynal, M.: The Combined Power of Conditions and Failure Detectors to Solve Asynchronous Set Agreement. In: Proc. 24th ACM Symposium on Principles of Distributed Computing (PODC'05), pp. 179–188. ACM Press, New York (2005)
9. Mostefaoui, A., Raynal, M.: *k*-Set Agreement with Limited Accuracy Failure Detectors. In: Proc. 19th ACM Symposium on Principles of Distributed Computing, pp. 143–152. ACM Press, New York (2000)
10. Mostefaoui, A., Raynal, M.: Randomized Set Agreement. In: Proc. 13th ACM Symposium on Parallel Algorithms and Architectures (SPAA'01), Hersonissos (Crete) pp. 291–297. ACM Press, New York (2001)
11. Perry, K.J., Toueg, S.: Distributed Agreement in the Presence of Processor and Communication Faults. IEEE Trans. Softw. Eng. **SE-12**(3), 477–482 (1986)
12. Raipin Parvedy, P., Raynal, M., Travers, C.: Early-stopping *k*-set agreement in synchronous systems prone to any number of process crashes. In: Proc. 8th Int'l Conference on Parallel Computing Technologies (PaCT'05). LNCS, vol. 3606, pp. 49–58. Springer, Berlin (2005)
13. Raipin Parvedy, P., Raynal, M., Travers, C.: Strongly-terminating early-stopping *k*-set agreement in synchronous systems with general omission failures. In: Proc. 13th Colloquium on Structural Information and Communication Complexity (SIROCCO'06). LNCS, vol. 4056, pp. 182–196. Springer, Berlin (2006)
14. Raynal, M., Travers, C.: Synchronous set agreement: a concise guided tour (including a new algorithm and a list of open problems). In: Proc. 12th Int'l IEEE Pacific Rim Dependable Computing Symposium (PRDC'2006), pp. 267–274. IEEE Society Computer Press, Los Alamitos (2006)
15. Saks, M., Zaharoglou, F.: Wait-Free *k*-Set Agreement is Impossible: The Topology of Public Knowledge. SIAM J. Comput. **29**(5), 1449–1483 (2000)

Set Cover with Almost Consecutive Ones
2004; Mecke, Wagner

MICHAEL DOM
Department of Mathematics and Computer Science,
University of Jena, Jena, Germany

Keywords and Synonyms

Hitting set

Problem Definition

The SET COVER problem has as input a set R of m items, a set C of n subsets of R and a weight function $w\colon C \to \mathbb{Q}$. The task is to choose a subset $C' \subseteq C$ of minimum weight whose union contains all items of R.

The sets R and C can be represented by an $m \times n$ binary matrix A that consists of a row for every item in R and a column for every subset of R in C, where an entry $a_{i,j}$ is 1 iff the ith item in R is part of the jth subset in C. Therefore, the SET COVER problem can be formulated as follows.

Input: An $m \times n$ binary matrix A and a weight function w on the columns of A.
Task: Select some columns of A with minimum weight such that the submatrix A' of A that is induced by these columns has at least one 1 in every row.

While SET COVER is NP-hard in general [4], it can be solved in polynomial time on instances whose columns can be permuted in such a way that in every row the ones appear consecutively, that is, on instances that have the *consecutive ones property (C1P)*.[1]

Motivated by problems arising from railway optimization, Mecke and Wagner [7] consider the case of SET COVER instances that have "almost the C1P". Having almost the C1P means that the corresponding matrices are similar to matrices that have been generated by starting with a matrix that has the C1P and replacing randomly a certain percentage of the 1's by 0's [7]. For Ruf and Schöbel [8], in contrast, having almost the C1P means that the average number of blocks of consecutive 1's per row is much smaller than the number of columns of the matrix. This entry will also mention some of their results.

[1]The C1P can be defined symmetrically for columns; this article focuses on rows. SET COVER on instances with the C1P can be solved in polynomial time, e. g., with a linear programming approach, because the corresponding coefficient matrices are totally unimodular (see [9]).

Notation

Given an instance (A, w) of SET COVER, let R denote the row set of A and C its column set. A column c_j *covers* a row r_i, denoted by $r_i \in c_j$, if $a_{i,j} = 1$.

A binary matrix has the *strong C1P* if (without any column permutation) the 1's appear consecutively in every row. A *block of consecutive 1's* is a maximal sequence of consecutive 1's in a row. It is possible to determine in linear time if a matrix has the C1P, and if so, to compute a column permutation that yields the strong C1P [2,3,6]. However, note that it is NP-hard to permute the columns of a binary matrix such that the number of blocks of consecutive 1's in the resulting matrix is minimized [1,4,5].

A *data reduction rule* transforms in polynomial time a given instance I of an optimization problem into an instance I' of the same problem such that $|I'| < |I|$ and the optimal solution for I' has the same value (e. g., weight) as the optimal solution for I. Given a set of data reduction rules, *to reduce* a problem instance means to repeatedly apply the rules until no rule is applicable; the resulting instance is called *reduced*.

Key Results

Data Reduction Rules

For SET COVER there exist well-known data reduction rules:

Row domination rule: If there are two rows $r_{i_1}, r_{i_2} \in R$
with $\forall c \in C\colon r_{i_1} \in c$ implies $r_{i_2} \in c$, then r_{i_2} is *dominated* by r_{i_1}. Remove row r_{i_2} from A.

Column domination rule: If there are two columns $c_{j_1}, c_{j_2} \in C$ with $w(c_{j_1}) \geq w(c_{j_2})$ and $\forall r \in R\colon r \in c_{j_1}$ implies $r \in c_{j_2}$, then c_{j_1} is *dominated* by c_{j_2}. Remove c_{j_1} from A.

In addition to these two rules, a column $c_{j_1} \in C$ can also be dominated by a subset $C' \subseteq C$ of the columns instead of a single column: If there is a subset $C' \subseteq C$ with $w(c_{j_1}) \geq \sum_{c \in C'} w(c)$ and $\forall r \in R\colon r \in c_{j_1}$ implies $(\exists c \in C'\colon r \in c)$, then remove c_{j_1} from A. Unfortunately, it is NP-hard to find a dominating subset C' for a given set c_{j_1}. Mecke and Wagner [7], therefore, present a restricted variant of this generalized column domination rule.

For every row $r \in R$, let $c_{\min}(r)$ be a column in C that covers r and has minimum weight under this property. For two columns $c_{j_1}, c_{j_2} \in C$, define $X(c_{j_1}, c_{j_2}) := \{c_{\min}(r) \mid r \in c_{j_1} \wedge r \notin c_{j_2}\}$. The new data reduction rule then reads as follows.

Advanced column domination rule: If there are two columns $c_{j_1}, c_{j_2} \in C$ and a row that is covered by both c_{j_1}

and c_{j_2}, and if $w(c_{j_1}) \geq w(c_{j_2}) + \sum_{c \in X(c_{j_1}, c_{j_2})} w(c)$, then c_{j_1} is *dominated* by $\{c_{j_2}\} \cup X(c_{j_1}, c_{j_2})$. Remove c_{j_1} from A.

Theorem 1 ([7]) *A matrix A can be reduced in $O(Nn)$ time with respect to the column domination rule, in $O(Nm)$ time with respect to the row domination rule, and in $O(Nmn)$ time with respect to all three data reduction rules described above, when N is the number of 1's in A.*

In the databases used by Ruf and Schöbel [8], matrices are represented by the column indices of the first and last 1's of its blocks of consecutive 1's. For such matrix representations, a fast data reduction rule is presented [8], which eliminates "unnecessary" columns and which, in the implementations, replaces the column domination rule. The new rule is faster than the column domination rule (a matrix can be reduced in $O(mn)$ time with respect to the new rule), but not as powerful: Reducing a matrix A with the new rule can result in a matrix that has more columns than the matrix resulting from reducing A with the column domination rule.

Algorithms

Mecke and Wagner [7] present an algorithm that solves SET COVER by enumerating all feasible solutions.

Given a row r_i of A, a *partial solution for the rows* $r_1, \ldots, r_i$ is a subset $C' \subseteq C$ of the columns of A such that for each row r_j with $j \in \{1, \ldots, i\}$ there is a column in C' that covers row r_j.

The main idea of the algorithm is to find an optimal solution by iterating over the rows of A and updating in every step a data structure S that keeps *all* partial solutions for the rows considered so far. More exactly, in every iteration step the algorithm considers the first row of A and updates the data structure S accordingly. Thereafter, the first row of A is deleted. The following code shows the algorithm.

```
1 Repeat $m$ times: {
2   for every partial solution $C'$ in $S$ that does not cover
                                          the first row of $A$: {
3       for every column $c$ of $A$ that covers the first row
                                                      of $A$: {
4           Add $\{c\} \cup C'$ to $S$; }
5       Delete $C'$ from $S$; }
6   Delete the first row of $A$; }
```

This straightforward enumerative algorithm could create a set S of exponential size. Therefore, the data reduction rules presented above are used to delete after each iteration step partial solutions that are not needed any more. To this end, a matrix B is associated with the set S, where every row corresponds to a row of A and every column corresponds to a partial solution in S—an entry $b_{i,j}$ of B is 1 iff the jth partial solution of B contains a column of A that covers the row r_i. The algorithm uses the matrix $C := \left(\begin{array}{c|c} A & B \\ \hline 0\ldots0 & 1\ldots1 \end{array}\right)$, which is updated together with S in every iteration step.[2] Line 6 of the code shown above is replaced by the following two lines:

```
6   Delete the first row of the matrix $C$;
7   Reduce the matrix $C$ and update $S$ accordingly; }
```

At the end of the algorithm, S contains exactly one solution, and this solution is optimal. Moreover, if the SET COVER instance is nicely structured, the algorithm has polynomial running time:

Theorem 2 ([7]) *If A has the strong C1P, is reduced, and its rows are sorted in lexicographic order, then the algorithm has a running time of $O(M^{3n})$ where M is the maximum number of 1's per row and per column.*

Theorem 3 ([7]) *If the distance between the first and the last 1 in every column is at most k, then at any time throughout the algorithm the number of columns in the matrix B is $O(2^{kn})$, and the running time is $O(2^{2k}kmn^2)$.*

Ruf and Schöbel [8] present a branch and bound algorithm for SET COVER instances that have a small average number of blocks of consecutive 1's per row.

The algorithm considers in each step a row r_i of the current matrix (which has been reduced with data reduction rules before) and branches into bl_i cases, where bl_i is the number of blocks of consecutive 1's in r_i. In each case, one block of consecutive 1's in row r_i is selected, and the 1's of all other blocks in this row are replaced by 0's. Thereafter, a lower and an upper bound on the weight of the solution for each resulting instance is computed. If a lower bound differs by a factor of more than $1 + \epsilon$, for a given constant ε, from the best upper bound achieved so far, the corresponding instance is subjected to further branchings. Finally, the best upper bound that was found is returned.

In each branching step, the bl_i instances that are newly generated are "closer" to have the (strong) C1P than the instance from which they descend. If an instance has the C1P, the lower and upper bound can easily be computed by exactly solving the problem. Otherwise, standard heuristics are used.

[2]The last row of C allows to distinguish the columns belonging to A from those belonging to B.

Applications

Set Cover instances occur e. g. in railway optimization, where the task is to determine where new railway stations should be built. Each row then corresponds to an existing settlement, and each column corresponds to a location on the existing trackage where a railway station could be build. A column c covers a row r, if the settlement corresponding to r lies within a given radius around the location corresponding to c.

If the railway network consisted of one straight line rail track only, the corresponding Set Cover instance would have the C1P; instances arising from real world data are close to have the C1P [7,8].

Experimental Results

Mecke and Wagner [7] make experiments on real-world instances as described in the Applications section and on instances that have been generated by starting with a matrix that has the C1P and replacing randomly a certain percentage of the 1's by 0's. The real-world data consists of a railway graph with 8200 nodes and 8700 edges, and 30 000 settlements. The generated instances consist of 50–50 000 rows with 10–200 1's per row. Up to 20% of the 1's are replaced by 0's.

In the real-world instances, the data reduction rules decrease the number of 1's to between 1% and 25% of the original number of 1's without and to between 0.2% and 2.5% with the advanced column reduction rule. In the case of generated instances that have the C1P, the number of 1's is decreased to about 2% without and to 0.5% with the advanced column reduction rule. In instances with 20% perturbation, the number of 1's is decreased to 67% without and to 20% with the advanced column reduction rule.

The enumerative algorithm has a running time that is almost linear for real-world instances and most generated instances. Only in the case of generated instances with 20% perturbation, the running time is quadratic.

Ruf and Schöbel [8] consider three instance types: real-world instances, instances arising from Steiner triple systems, and randomly generated instances. The latter have a size of 100×100 and contain either 1–5 blocks of consecutive 1's in each row, each one consisting of between one and nine 1's, or they are generated with a probability of 3% or 5% for any entry to be 1.

The data reduction rules used by Ruf and Schöbel turn out to be powerful for the real-world instances (reducing the matrix size from about 1100×3100 to 100×800 in average), whereas for all other instance types the sizes could not be reduced noticeably.

The branch and bound algorithm could solve almost all real-world instances up to optimality within a time of less than a second up to one hour. In all cases where an optimal solution has been found, the first generated subproblem had already provided a lower bound equal to the weight of the optimal solution.

Cross References

► Greedy Set-Cover Algorithms

Recommended Reading

1. Atkins, J.E., Middendorf, M.: On physical mapping and the consecutive ones property for sparse matrices. Discret. Appl. Math. **71**(1–3), 23–40 (1996)
2. Booth, K.S., Lueker, G.S.: Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. J. Comput. Syst. Sci. **13**, 335–379 (1976)
3. Fulkerson, D.R., Gross, O.A.: Incidence matrices and interval graphs. Pac. J. Math. **15**(3), 835–855 (1965)
4. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, New York (1979)
5. Goldberg, P.W., Golumbic, M.C., Kaplan, H., Shamir, R.: Four strikes against physical mapping of DNA. J. Comput. Biol. **2**(1), 139–152 (1995)
6. Hsu, W.L., McConnell, R.M.: PC trees and circular-ones arrangements. Theor. Comput. Sci. **296**(1), 99–116 (2003)
7. Mecke, S., Wagner, D.: Solving geometric covering problems by data reduction. In: Proceedings of the 12th Annual European Symposium on Algorithms (ESA '04). LNCS, vol. 3221, pp. 760–771. Springer, Berlin (2004)
8. Ruf, N., Schöbel, A.: Set covering with almost consecutive ones property. Discret. Optim. **1**(2), 215–228 (2004)
9. Schrijver, A.: Theory of Linear and Integer Programming. Wiley, Chichester (1986)

Shortest Elapsed Time First Scheduling

2003; Bansal, Pruhs

Nikhil Bansal
IBM Research, IBM, Yorktown Heights, NY, USA

Keywords and Synonyms

Sojourn time; Response time; Scheduling with unknown job sizes; MLF algorithm; Feedback Queues

Problem Definition

The problem is concerned with scheduling dynamically arriving jobs in the scenario when the processing requirements of jobs are unknown to the scheduler. The lack of knowledge of how long a job will take to execute is a particularly attractive assumption in real systems where such information might be difficult or impossible to obtain. The goal is to schedule jobs to provide good quality of service to the users. In particular the goal is to design algorithms that have good average performance and are also fair in the sense that no subset of users experiences substantially worse performance than others.

Notations

Let $\mathcal{J} = \{1, 2, \ldots, n\}$ denote the set of jobs in the input instance. Each job j is characterized by its release time r_j and its processing requirement p_j. In the online setting, job j is revealed to the scheduler only at time r_j. A further restriction is the *non-clairvoyant* setting, where only the existence of job j is revealed at r_j, in particular the scheduler does not know p_j until the job meets its processing requirement and leaves the system. Given a schedule, the completion time c_j of a job is the earliest time at which job j receives p_j amount of service. The flow time f_j of j is defined as $c_j - r_j$. The stretch of a job is defined the ratio of its flow time divided by its size. Stretch is also referred to as normalized flow time or slowdown, and is a natural measure of fairness as it measures the waiting time of a job per unit of service received. A schedule is said to be preemptive, if a job can be interrupted arbitrarily, and its execution can be resumed later from the point of interruption without any penalty. It is well known that preemption is necessary to obtain reasonable guarantees for flow time even in the offline setting [5].

Recall that the online Shortest Remaining Processing Time (SRPT) algorithm, that at any time works on the job with the least remaining processing, is optimum for minimizing average flow time. However, a common critique of SRPT is that it may lead to starvation of jobs, where some jobs may be delayed indefinitely. For example, consider the sequence where a job of size 3 arrives at time $t = 0$, and one job of size 1 arrives every unit of time starting $t = 1$ for a long time. Under SRPT the size 3 job will be delayed until the size 1 jobs stop arriving. On the other hand, if the goal is to minimize the maximum flow time, then it is easily seen that First in First out (FIFO) is the optimum algorithm. However, FIFO can perform very poorly with respect to average flow time (for example, many small jobs could be stuck behind a very large job that arrived just earlier). A natural way to balance both the average and worst case performance is to consider the ℓ_p norms of flow time and stretch, where the ℓ_p norm of the sequence $x_1, \ldots, x_n$ is defined as $(\sum_i x_i^p)^{1/p}$.

The Shortest Elapsed Time First (SETF) is a non-clairvoyant algorithm that at any time works on the job that has received the least amount of service thus far. This is a natural way to favor short jobs given the lack of knowledge of job sizes. In fact, SETF is the continuous version of the Multi-Level Feedback (MLF) algorithm. Unfortunately, SETF (or any other deterministic non-clairvoyant algorithm) performs poorly in the framework of competitive analysis, where an algorithm is called c-competitive if for every input instance, its performance is no worse than c times that of the optimum offline (clairvoyant) solution for that instance [7]. However, competitive analysis can be overly pessimistic in its guarantee. A way around this problem was proposed by Kalyanasundaram and Pruhs [6] who allowed the online scheduler a slightly faster processor to make up for its lack of knowledge of future arrivals and job sizes. Formally, an algorithm *Alg* is said to be s-speed, c-speed competitive where c is worst case ratio over all instance I, of $Alg_s(I)/Opt_1(I)$, where Alg_s is the value of solution produced by *Alg* when given an s speed processor, and Opt_1 is the optimum value using a speed 1 processor. Typically the most interesting results are those where c is small and $s = (1 + \epsilon)$ for any arbitrary $\epsilon > 0$.

Key Results

In their seminal paper [6], Kalyanasundaram and Pruhs showed the following.

Theorem 1 **([6])** *SETF is a $(1 + \epsilon)$-speed, $(1 + 1/\epsilon)$-competitive non-clairvoyant algorithm for minimizing the average flow time on a single machine with preemptions.*

For minimizing the average stretch, Muthukrishnan, Rajaraman, Shaheen and Gehrke [8] considered the clairvoyant setting and showed that SRPT is 2-competitive for a single machine and 14 competitive for multiple machines. The non-clairvoyant setting was consider by Bansal, Dhamdhere, Konemann and Sinha [1]. They showed that

Theorem 2 **([1])** *SETF is a $(1 + \epsilon)$-speed, $O(\log^2 P)$-competitive for minimizing average stretch, where P is the ratio of the maximum to minimum job size. On the other hand, even with O(1)-speed, any non-clairvoyant algorithm is at least $\Omega(\log P)$-competitive. Interestingly, in terms of n, any non-clairvoyant algorithm must be $\Omega(n)$-competitive even with O(1)-speedup. Moreover, SETF is O(n) competitive (even without extra speedup).*

For the special case when all jobs arrive at time 0, SETF is optimum up to constant factors. It is $O(\log P)$-competitive (without any extra speedup). Moreover, any *non-clairvoyant must be $\Omega(\log P)$ competitive even with factor O(1) speedup.*

The key idea of the above result was a connection between SETF and SRPT. First, at the expense of $(1+\epsilon)$-speedup it can be seen that SETF is no worse than MLF where the thresholds are powers of $(1+\epsilon)$. Second, the behavior of MLF on an instance I can be related to the behavior of Shortest Job First (SJF) algorithm on another instance I' that is obtained from I by dividing each job into logarithmically many jobs with geometrically increasing sizes. Finally, the performance of SJF is related to SRPT using another $(1+\epsilon)$ factor speedup.

Bansal and Pruhs [2] considered the problem of minimizing the ℓ_p norms of flow time and stretch on a single machine. They showed the following.

Theorem 3 ([2]) *In the clairvoyant setting, SRPT and SJF are $(1+\epsilon)$-speed, $O(1/\epsilon)$-competitive for minimizing the ℓ_p norms of both flowtime and stretch. On the other hand, for $1 < p < \infty$, no online algorithm (possibly clairvoyant) can be O(1) competitive for minimizing ℓ_p norms of stretch or flow time without speedup. In particular, any randomized online algorithm is at least $\Omega(n^{(p-1)/3p^2})$-competitive for ℓ_p norms of stretch, and is at least $\Omega(n^{(p-1)/p(3p-1)})$-competitive for ℓ_p norms of flow time.*

The above lower bounds are somewhat surprising, since SRPT and FIFO are optimum for the case $p = 1$ and $p = \infty$ for flow time.

Bansal and Pruhs [2] also consider the non-clairvoyant case.

Theorem 4 ([2]) *In the non-clairvoyant setting, SETF is $(1+\epsilon)$-speed, $O(1/\epsilon^{2+2/p})$-competitive for minimizing the ℓ_p norms of flow time. For minimizing ℓ_p norms of stretch, SETF is $(1+\epsilon)$-speed, $O(1/\epsilon^{3+1/p} \cdot \log^{1+1/p} P)$-competitive.*

Finally, Bansal and Pruhs also consider Round Robin (RR) or Processor Sharing that at any time splits the processor equally among the unfinished jobs. RR is considered to be an ideal fair strategy since it treats all unfinished jobs equally. However, they show that

Theorem 5 *For any $p \geq 1$, there is an $\epsilon > 0$ such that even with a $(1+\epsilon)$ times faster processor, RR is not $n^{o(1)}$ competitive for minimizing the ℓ_p norms of flow time. In particular, for $\epsilon < 1/2p$, RR is $(1+\epsilon)$-speed, $\Omega(n^{(1-2\epsilon p)/p})$-competitive. For ℓ_p norms of stretch, RR is $\Omega(n)$ competitive as is in fact any randomized non-clairvoyant algorithm.*

The results above have been extended in a couple of directions. Bansal and Pruhs [3] extend these results to *weighted* ℓ_p norms of flow time and stretch. Chekuri, Khanna, Kumar and Goel [4] have extended these results to the multiple machines case. Their algorithms are particularly elegant: Each job is assigned to some machine at random and all jobs at a particular machine are processed using SRPT or SETF (as applicable).

Applications

SETF and its variants such as MLF are widely used in operating systems [9,10]. Note that SETF is not really practical since each job could be preempted infinitely often. However, variants of SETF with fewer preemptions are quite popular.

Open Problems

It would be interesting to explore other notions of fairness in the dynamic scheduling setting. In particular, it would be interesting to consider algorithms that are both fair and have a good average performance.

An immediate open problem is whether the gap between $O(\log^2 P)$ and $\Omega(\log P)$ can be closed for minimizing the average stretch in the non-clairvoyant setting.

Cross References

▶ Flow Time Minimization
▶ Minimum Flow Time
▶ Multi-level Feedback Queues

Recommended Reading

1. Bansal, N., Dhamdhere, K., Könemann, J., Sinha, A.: Non-Clairvoyant Scheduling for Minimizing Mean Slowdown. Algorithmica **40**(4), 305–318 (2004)
2. Bansal, N., Pruhs, K.: Server scheduling in the Lp norm: a rising tide lifts all boat. In: Symposium on Theory of Computing, STOC, pp. 242–250 (2003)
3. Bansal, N., Pruhs, K.: Server scheduling in the weighted Lp norm. In: LATIN, pp. 434–443 (2004)
4. Chekuri, C., Goel, A., Khanna, S., Kumar, A.: Multi-processor scheduling to minimize flow time with epsilon resource augmentation. In: Symposium on Theory of Computing, STOC, pp. 363–372 (2004)
5. Kellerer, H., Tautenhahn, T., Woeginger, G.J.: Approximability and Nonapproximability Results for Minimizing Total Flow Time on a Single Machine. SIAM J. Comput. **28**(4), 1155–1166 (1999)
6. Kalyanasundaram, B., Pruhs, K.: Speed is as powerful as clairvoyance. J. ACM **47**(4), 617–643 (2000)
7. Motwani, R., Phillips, S., Torng, E.: Non-Clairvoyant Scheduling. Theor. Comput. Sci. **130**(1), 17–47 (1994)
8. Muthukrishnan, S., Rajaraman, R., Shaheen, A., Gehrke, J.: Online Scheduling to Minimize Average Stretch. SIAM J. Comput. **34**(2), 433–452 (2004)
9. Nutt, G.: Operating System Projects Using Windows NT. Addison Wesley, Reading (1999)

10. Tanenbaum, A.S.: Modern Operating Systems. Prentice-Hall Inc., Englewood Cliffs (1992)

Shortest Path

► Algorithms for Spanners in Weighted Graphs
► All Pairs Shortest Paths via Matrix Multiplication
► Maximum-scoring Segment with Length Restrictions
► Routing in Road Networks with Transit Nodes

Shortest Paths Approaches for Timetable Information

2004; Pyrga, Schulz, Wagner, Zaroliagis

RIKO JACOB
Institute of Computer Science,
Technical University of Munich, Munich, Germany

Keywords and Synonyms

Passenger information system; Timetable lookup; Journey planner; Trip planner

Problem Definition

Consider the route-planning task for passengers of scheduled public transportation. Here, the running example is that of a train system, but the discussion applies equally to bus, light-rail and similar systems. More precisely, the task is to construct a timetable information system that, based upon the detailed schedules of all trains, provides passengers with good itineraries, including the transfer between different trains.

Solutions to this problem consist of a model of the situation (e. g. can queries specify a limit on the number of transfers?), an algorithmic approach, its mathematical analysis (does it always return the best solution? Is it guaranteed to work fast in all settings?), and an evaluation in the real world (Can travelers actually use the produced itineraries? Is an implementation fast enough on current computers and real data?).

Key Results

The problem is discussed in detail in a recent survey article [6].

Modeling

In a simplistic model, it is assumed that a transfer between trains does not take time. A more realistic model specifies a certain minimum transfer time per station. Furthermore, the objective of the optimization problem needs to be defined. Should the itinerary be as fast as possible, or as cheap as possible, or induce the least possible transfers? There are different ways to resolve this as surveyed in [6], all originating in multi-objective optimization, like resource constraints or Pareto-optimal solutions. From a practical point of view, the preferences of a traveler are usually difficult to model mathematically, and one might want to let the user choose the best option among a set of reasonable itineraries himself. For example, one can compute all itineraries that are not inferior to some other itinerary in all considered aspects. As it turns out, in real timetables the number of such itineraries is not too big, such that this approach is computationally feasible and useful for the traveler [5]. Additionally, the fare structure of most railways is fairly complicated [4], mainly because fares usually are not additive, i. e., are not the sum of fares of the parts of a trip.

Algorithmic Models

The current literature establishes two main ideas how to transform the situation into a shortest path problem on a graph. As an example, consider the simplistic modeling where transfer takes no time, and where queries specify starting time and station to ask for an itinerary that achieves the earliest arrival time at the destination.

In the time-expanded model [11], every arrival and departure event of the timetable is a vertex of the directed graph. The arcs of the graph represent consecutive events at one station, and direct train connections. The length of an arc is given by the time difference of its end vertices. Let s be the vertex at the source station whose time is directly after the starting time. Now, a shortest path from s to any vertex of the destination station is an optimal itinerary.

In the time-dependent model [3,7,9,10], the vertices model stations, and the arcs stand for the existence of a direct (non-stop) train connection. Instead of edge length, the arcs are labeled with edge-traversal functions that give the arrival time at the end of the arc in dependence on the time a passenger starts at the beginning of the arc, reflecting the times when trains actually run. To solve this time-dependent shortest path problem, a modification of Dijkstra's algorithm can be used. Further exploiting the structure of this situation, the graph can be represented in a way that allows constant time evaluation of the link traversal functions [3]. To cope with more realistic transfer models, a more complicated graph can be used.

Additionally, many of the speed-up techniques for shortest path computations can be applied to the resulting graph queries.

Applications

The main application are timetable information systems for scheduled transit (buses, trains, etc.). This extends to route planning where trips in such systems are allowed, as for example in the setting of fine-grained traffic simulation to compute fastest itineraries [2].

Open Problems

Improve computation speed, in particular for fully integrated timetables and the multi-criteria case. Extend the problem to the dynamic case, where the current real situation is reflected, i. e., delayed or canceled trains, and otherwise temporarily changed timetables are reflected.

Experimental Results

In the cited literature, experimental results usually are part of the contribution [2,4,5,6,7,8,9,10,11]. The time-dependent approach can be significantly faster than the time-expanded approach. In particular for the simplistic models speed-ups in the range 10–45 are observed [8,10]. For more detailed models, the performance of the two approaches becomes comparable [6].

Cross References

► Implementation Challenge for Shortest Paths
► Routing in Road Networks with Transit Nodes
► Single-Source Shortest Paths

Acknowledgments

I want to thank Matthias Müller-Hannemann, Dorothea Wagner, and Christos Zaroliagis for helpful comments on an earlier draft of this entry.

Recommended Reading

1. Gerards, B., Marchetti-Spaccamela, A. (eds.): Proceedings of the 3rd Workshop on Algorithmic Methods and Models for Optimization of Railways (ATMOS'03) 2003. Electronic Notes in Theoretical Computer Science, vol. 92. Elsevier (2004)
2. Barrett, C.L., Bisset, K., Jacob, R., Konjevod, G., Marathe, M.V.: Classical and contemporary shortest path problems in road networks: Implementation and experimental analysis of the TRANSIMS router. In: Algorithms – ESA 2002: 10th Annual European Symposium, Rome, Italy, 17–21 September 2002. Lecture Notes Computer Science, vol. 2461, pp. 126–138. Springer, Berlin (2002)
3. Brodal, G.S., Jacob, R.: Time-dependent networks as models to achieve fast exact time-table queries. In: Proceedings of the 3rd Workshop on Algorithmic Methods and Models for Optimization of Railways (ATMOS'03), 2003, [1], pp. 3–15
4. Müller-Hannemann, M., Schnee, M.: Paying less for train connections with MOTIS. In: Kroon, L.G., Möhring, R.H. (eds.) Proceedings of the 5th Workshop on Algorithmic Methods and Models for Optimization of Railways (ATMOS'05), Dagstuhl, Germany, Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany 2006. Dagstuhl Seminar Proceedings, no. 06901
5. Müller-Hannemann, M., Schnee, M.: Finding all attractive train connections by multi-criteria pareto search. In: Geraets, F., Kroon, L.G., Schöbel, A., Wagner, D., Zaroliagis, C.D. (eds.) Algorithmic Methods for Railway Optimization, International Dagstuhl Workshop, Dagstuhl Castle, Germany, June 20–25, 2004, 4th International Workshop, ATMOS 2004, Bergen, September 16–17, 2004, Revised Selected Papers, Lecture Notes in Computer Science, vol. 4359, pp. 246–263. Springer, Berlin (2007)
6. Müller-Hannemann, M., Schulz, F., Wagner, D., Zaroliagis, C.D.: Timetable information: Models and algorithms. In: Geraets, F., Kroon, L.G., Schöbel, A., Wagner, D., Zaroliagis, C.D. (eds.) Algorithmic Methods for Railway Optimization, International Dagstuhl Workshop, Dagstuhl Castle, Germany, June 20–25, 2004, 4th International Workshop, ATMOS 2004, Bergen, September 16–17, 2004, Revised Selected Papers, Lecture Notes in Computer Science, vol. 4359, pp. 67–90. Springer (2007)
7. Nachtigall, K.: Time depending shortest-path problems with applications to railway networks. Eur. J. Oper. Res. **83**, 154–166 (1995)
8. Pyrga, E., Schulz, F., Wagner, D., Zaroliagis, C.: Experimental comparison of shortest path approaches for timetable information. In: Proceedings 6th Workshop on Algorithm Engineering and Experiments (ALENEX), Society for Industrial and Applied Mathematics, 2004, pp. 88–99
9. Pyrga, E., Schulz, F., Wagner, D., Zaroliagis, C.: Towards realistic modeling of time-table information through the time-dependent approach. In: Proceedings of the 3rd Workshop on Algorithmic Methods and Models for Optimization of Railways (ATMOS'03), 2003, [1], pp. 85–103
10. Pyrga, E., Schulz, F., Wagner, D., Zaroliagis, C.: Efficient models for timetable information in public transportation systems. J. Exp. Algorithmics **12**, 2.4 (2007)
11. Schulz, F., Wagner, D., Weihe, K.: Dijkstra's algorithm on-line: An empirical case study from public railroad transport. J. Exp. Algorithmics **5** 1–23 (2000)

Shortest Paths in Planar Graphs with Negative Weight Edges

2001; Fakcharoenphol, Rao

JITTAT FAKCHAROENPHOL[1], SATISH RAO[2]
[1] Department of Computer Engineering, Kasetsart University, Bangkok, Thailand
[2] Department of Computer Science, University of California at Berkeley, Berkeley, CA, USA

Keywords and Synonyms

Shortest paths in planar graphs with general arc weights; Shortest paths in planar graphs with arbitrary arc weights

Problem Definition

This problem is to find shortest paths in planar graphs with general edge weights. It is known that shortest paths exist only in graphs that contain no negative weight cycles. Therefore, algorithms that work in this case must deal with the presence of negative cycles, i. e., they must be able to detect negative cycles.

In general graphs, the best known algorithm, the Bellman-Ford algorithm, runs in time $O(mn)$ on graphs with n nodes and m edges, while algorithms on graphs with no negative weight edges run much faster. For example, Dijkstra's algorithm implemented with the Fibonacchi heap runs in time $O(m + n \log n)$, and, in case of integer weights Thorup's algorithm runs in linear time. Goldberg [5] also presented an $O(m\sqrt{n} \log L)$-time algorithm where L denotes the absolute value of the most negative edge weights. Note that his algorithm is weakly polynomial.

Notations

Given a directed graph $G = (V, E)$ and a weight function $w\colon E \to \mathbb{R}$ on its directed edges, a *distance labeling* for a source node s is a function $d\colon V \to \mathbb{R}$ such that $d(v)$ is the minimum length over all s-to-v paths, where the *length of path P* is $\sum_{e \in P} w(e)$.

Problem 1 (Single-Source-Shortest-Path)
INPUT: *A directed graph* $G = (V, E)$, *weight function* $w\colon E \to \mathbb{R}$, *source node* $s \in V$.
OUTPUT: *If G does not contain negative length cycles, output a distance labeling d for source node s. Otherwise, report that the graph contains some negative length cycle.*

The algorithm by Fakcharoenphol and Rao [4] deals with the case when G is planar. They gave an $O(n \log^3 n)$-time algorithm, improving on an $O(n^{3/2})$-time algorithm by Lipton, Rose, and Tarjan [9] and an $O(n^{4/3} \log nL)$-time algorithm by Henzinger, Klein, Rao, and Subramanian [6].

Their algorithm, as in all previous algorithms, uses a recursive decomposition and constructs a data structure called a dense distance graph, which shall be defined next.

A *decomposition* of a graph is a set of subsets $P_1, P_2, \ldots, P_k$ (not necessarily disjoint) such that the union of all the sets is V and for all $e = (u, v) \in E$, there is a unique P_i that contains e. A node v is a *border node* of a set P_i if $v \in P_i$ and there exists an edge $e = (v, x)$ where $x \notin P_i$. The subgraph induced on a subset P_i is referred to as a *piece* of the decomposition.

The algorithm works with a *recursive decomposition* where at each level, a piece with n nodes and r border nodes is divided into two subpieces such that each subpiece has no more than $2n/3$ nodes and at most $2r/3 + c\sqrt{n}$ border nodes, for some constant c. In this recursive context, a border node of a subpiece is defined to be any border node of the original piece or any new border node introduced by the decomposition of the current piece.

With this recursive decomposition, the *level of a decomposition* can be defined in the natural way, with the entire graph being the only piece in the level 0 decomposition, the pieces of the decomposition of the entire graph being the level 1 pieces in the decomposition, and so on.

For each piece of the decomposition, the all-pair shortest path distances between all its border nodes along paths that lie entirely inside the piece are recursively computed. These all-pair distances form the edge set of a non-planar graph representing shortest paths between border nodes. The dense distance graph of the planar graph is the union of these graphs over all the levels.

Using the dense distance graph, the shortest distance queries between pairs of nodes can be answered.

Problem 2 (Shortest-Path-Distance-Data-Structure)
INPUT: *A directed graph* $G = (V, E)$, *weight function* $w\colon E \to \mathbb{R}$, *source node* $s \in V$.
OUTPUT: *If G does not contain negative length cycles, output a data structure that support distance queries between pairs of nodes. Otherwise, report that the graph contains some negative length cycle.*

The algorithm of Fakcharoenphol and Rao relies heavily on planarity, i. e., it exploits properties regarding how shortest paths on each piece intersect. Therefore, unlike previous algorithms that require only that the graph can be recursively decomposed with small numbers of border nodes [10], their algorithm also requires that each piece has a nice embedding.

Given an embedding of the piece, a *hole* is a bounded face where all adjacent nodes are border nodes. Ideally, one would hope that there is a planar embedding of any piece in the recursive decomposition where all the border nodes are on a single face and are circularly ordered, i. e., there is no holes in each piece. Although this is not always true, the algorithm works with any decomposition with a constant number of holes in each piece. This decomposition can be found in $O(n \log n)$ time using the simple cycle separator algorithm by Miller [12].

Key Results

Theorem 1 *Given a recursive decomposition of a planar graph such that each piece of the decomposition contains at most a constant number of holes, there is an algorithm that constructs the dense distance graph is $O(n \log^3 n)$ time.*

Given the procedure that constructs the dense distance graph, the shortest paths from a source s can be computed by first adding s as a border node in every piece of the decomposition, computing the dense distance graph, and then extending the distances into all internal nodes on every piece. This can be done in time $O(n \log^3 n)$.

Theorem 2 *The single-source shortest path problem for an n-node planar graph with negative weight edges can be solved in time $O(n \log^3 n)$.*

The dense distance graph can be used to answer distance queries between pairs of nodes.

Theorem 3 *Given the dense distance graph, the shortest distance between any pair of nodes can be found in $O(\sqrt{n} \log^2 n)$ time.*

It can also be used as a dynamic data structure that answers shortest path queries and allows edge cost updates.

Theorem 4 *For planar graphs with only non-negative weight edges, there is a dynamic data structure that supports distance queries and update operations that change edge weights in amortized $O(n^{2/3} \log^{7/3} n)$ time per operation. For planar graph with negative weight edges, there is a dynamic data structures that supports the same set of operations in amortized $O(n^{4/5} \log^{13/5} n)$ time per operation.*

Note that the dynamic data structure does not support edge insertions and deletions, since these operations might destroy the recursive decomposition.

Applications

The shortest path problem has long been studied and continues to find applications in diverse areas. There are a many problems that reduce to the shortest path problem where negative weight edges are required, for example the minimum-mean length directed circuit. For planar graphs, the problem has wide application even when the underlying graph is a grid. For example, there are recent image segmentation approaches that use negative cycle detection [2,3]. Some of other applications for planar graphs include separator algorithms [13] and multi-source multi-sink flow algorithms [11].

Open Problems

Klein [8] gives a technique that improves the running time of the construction of the dense distance graph to $O(n \log^2 n)$ when all edge weights are non-negative; this also reduces the amortized running time for the dynamic case down to $O(n^{2/3} \log^{5/3} n)$. Also, for planar graphs with no negative weight edges, Cabello [1] gives a faster algorithm for computing the shortest distances between k pairs of nodes. However, the problem for improving the bound of $O(n \log^3 n)$ for finding shortest paths in planar graphs with general edge weights remains opened.

It is not known how to handle edge insertions and deletions in the dynamic data structure. A new data structure might be needed instead of the dense distance graph, because the dense distance graph is determined by the decomposition.

Cross References

- ▶ All Pairs Shortest Paths in Sparse Graphs
- ▶ All Pairs Shortest Paths via Matrix Multiplication
- ▶ Approximation Schemes for Planar Graph Problems
- ▶ Decremental All-Pairs Shortest Paths
- ▶ Fully Dynamic All Pairs Shortest Paths
- ▶ Implementation Challenge for Shortest Paths
- ▶ Negative Cycles in Weighted Digraphs
- ▶ Planarity Testing
- ▶ Shortest Paths Approaches for Timetable Information
- ▶ Single-Source Shortest Paths

Recommended Reading

1. Cabello, S.: Many distances in planar graphs. In: SODA '06: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm, pp. 1213–1220. ACM Press, New York (2006)
2. Cox, I.J., Rao, S. B., Zhong, Y.: 'Ratio Regions': A Technique for Image Segmentation. In: Proceedings International Conference on Pattern Recognition, IEEE, pp. 557–564, August (1996)
3. Geiger, L.C.D., Gupta, A., Vlontzos, J.: Dynamic programming for detecting, tracking and matching elastic contours. IEEE Trans. On Pattern Analysis and Machine Intelligence (1995)
4. Fakcharoenphol, J., Rao, S.: Planar graphs, negative weight edges, shortest paths, and near linear time. J. Comput. Syst. Sci. **72**, 868–889 (2006)
5. Goldberg, A.V.: Scaling algorithms for the shortest path problem. SIAM J. Comput. **21**, 140–150 (1992)
6. Henzinger, M.R., Klein, P.N., Rao, S., Subramanian, S.: Faster Shortest-Path Algorithms for Planar Graphs. J. Comput. Syst. Sci. **55**, 3–23 (1997)
7. Johnson, D.: Efficient algorithms for shortest paths in sparse networks. J. Assoc. Comput. Mach. **24**, 1–13 (1977)
8. Klein, P.N.: Multiple-source shortest paths in planar graphs. In: Proceedings, 16th ACM-SIAM Symposium on Discrete Algorithms, pp. 146–155 (2005)

9. Lipton, R., Rose, D., Tarjan, R.E.: Generalized nested dissection. SIAM. J. Numer. Anal. **16**, 346–358 (1979)
10. Lipton, R.J., Tarjan, R.E.: A separator theorem for planar graphs. SIAM. J. Appl. Math. **36**, 177–189 (1979)
11. Miller, G., Naor, J.: Flow in planar graphs with multiple sources and sinks. SIAM J. Comput. **24**, 1002–1017 (1995)
12. Miller, G.L.: Finding small simple cycle separators for 2-connected planar graphs. J. Comput. Syst. Sci. **32**, 265–279 (1986)
13. Rao, S.B.: Faster algorithms for finding small edge cuts in planar graphs (extended abstract). In: Proceedings of the Twenty-Fourth Annual ACM Symposium on the Theory of Computing, pp. 229–240, May (1992)
14. Thorup, M.: Compact oracles for reachability and approximate distances in planar digraphs. J. ACM **51**, 993–1024 (2004)

Shortest Route

► All Pairs Shortest Paths in Sparse Graphs
► Rectilinear Steiner Tree
► Single-Source Shortest Paths

Shortest Vector Problem

1982; Lenstra, Lenstra, Lovasz

DANIELE MICCIANCIO
Department of Computer Science, University of California, San Diego, La Jolla, CA, USA

Keywords and Synonyms

Lattice basis reduction; LLL algorithm; Closest vector problem; Nearest vector problem; Minimum distance problem

Problem Definition

A *point lattice* is the set of all integer linear combinations

$$\mathcal{L}(\mathbf{b}_1,\ldots,\mathbf{b}_n)=\left\{\sum_{i=1}^{n}x_i\mathbf{b}_i : x_1,\ldots,x_n\in\mathbb{Z}\right\}$$

of n linearly independent vectors $\mathbf{b}_1,\ldots,\mathbf{b}_n\in\mathbb{R}^m$ in m-dimensional Euclidean space. For computational purposes, the lattice vectors $\mathbf{b}_1,\ldots,\mathbf{b}_n$ are often assumed to have integer (or rational) entries, so that the lattice can be represented by an integer matrix $\mathbf{B}=[\mathbf{b}_1,\ldots,\mathbf{b}_n]\in\mathbb{Z}^{m\times n}$ (called *basis*) having the generating vectors as columns. Using matrix notation, lattice points in $\mathcal{L}(\mathbf{B})$ can be conveniently represented as $\mathbf{Bx}$ where $\mathbf{x}$ is an integer vector. The integers m and n are called the *dimension* and *rank* of the lattice respectively. Notice that any lattice admits multiple bases, but they all have the same rank and dimension.

The main computational problems on lattices are the *Shortest Vector Problem*, which asks to find the shortest nonzero vector in a given lattice, and the *Closest Vector Problem*, which asks to find the lattice point closest to a given target. Both problems can be defined with respect to any norm, but the Euclidean norm $\|\mathbf{v}\|=\sqrt{\sum_i v_i^2}$ is the most common. Other norms typically found in computer science applications are the ℓ_1 norm $\|\mathbf{v}\|_1=\sum_i|v_i|$ and the *max* norm $\|\mathbf{v}\|_\infty=\max_i|v_i|$. This entry focuses on the Euclidean norm.

Since no efficient algorithm is known to solve SVP and CVP exactly in arbitrary high dimension, the problems are usually defined in their approximation version, where the approximation factor $\gamma\geq 1$ can be a function of the dimension or rank of the lattice.

Definition 1 (Shortest Vector Problem, SVP$_\gamma$) Given a lattice $\mathcal{L}(\mathbf{B})$, find a nonzero lattice vector $\mathbf{Bx}$ (where $\mathbf{x}\in\mathbb{Z}^n\setminus\{\mathbf{0}\}$) such that $\|\mathbf{Bx}\|\leq\gamma\cdot\|\mathbf{By}\|$ for any $\mathbf{y}\in\mathbb{Z}^n\setminus\{\mathbf{0}\}$.

Definition 2 (Closest Vector Problem, CVP$_\gamma$) Given a lattice $\mathcal{L}(\mathbf{B})$ and a target point $\mathbf{t}$, find a lattice vector $\mathbf{Bx}$ (where $\mathbf{x}\in\mathbb{Z}^n$) such that $\|\mathbf{Bx}-\mathbf{t}\|\leq\gamma\cdot\|\mathbf{By}-\mathbf{t}\|$ for any $\mathbf{y}\in\mathbb{Z}^n$.

Lattices have been investigated by mathematicians for centuries in the equivalent language of quadratic forms, and are the main object of study in the *geometry of numbers*, a field initiated by Minkowski as a bridge between geometry and number theory. For a mathematical introduction to lattices see [3]. The reader is referred to [6,12] for an introduction to lattices with an emphasis on computational and algorithmic issues.

Key Results

The problem of finding an efficient (polynomial time) solution to SVP$_\gamma$ for lattices in arbitrary dimension was first solved by the celebrated *lattice reduction* algorithm of Lenstra, Lenstra and Lovász [11], commonly known as the *LLL* algorithm.

Theorem 1 *There is a polynomial time algorithm to solve* SVP$_\gamma$ *for* $\gamma=(2/\sqrt{3})^n$, *where n is the rank of the input lattice.*

The LLL algorithm achieves more than just finding a relatively short lattice vector: it finds a so-called *reduced basis* for the input lattice, i. e., an entire basis of relatively short lattice vectors. Shortly after the discovery of the LLL algorithm, Babai [2] showed that reduced bases can be used to efficiently solve CVP$_\gamma$ as well within similar approximation factors.

Corollary 1 *There is a polynomial time algorithm to solve* CVP_γ *for* $\gamma = O(2/\sqrt{3})^n$*, where n is the rank of the input lattice.*

The reader is referred to the original papers [2,11] and [12, chap. 2] for details. Introductory presentations of the LLL algorithm can also be found in many other texts, e. g., [5, chap. 16] and [15, chap. 27]. It is interesting to note that CVP is at least as hard as SVP (see [12, chap 2]) in the sense that any algorithm that solves CVP_γ can be efficiently adapted to solve SVP_γ within the same approximation factor.

Both SVP_γ and CVP_γ are known to be NP-hard in their exact ($\gamma = 1$) or even approximate versions for small values of γ, e. g., constant γ independent of the dimension. (See [13, chaps. 3 and 4] and [4,10] for the most recent results.) So, no efficient algorithm is likely to exist to solve the problems exactly in arbitrary dimension. For any fixed dimension n, both SVP and CVP can be solved exactly in polynomial time using an algorithm of Kannan [9]. However, the dependency of the running time on the lattice dimension is $n^{O(n)}$. Using randomization, exact SVP can be solved probabilistically in $2^{O(n)}$ time and space using the *sieving* algorithm of Ajtai, Kumar and Sivakumar [1].

As for approximate solutions, the LLL lattice reduction algorithm has been improved both in terms of running time and approximation guarantee. (See [14] and references therein.) Currently, the best (randomized) polynomial time approximation algorithm achieves approximation factor $\gamma = 2^{O(n \log\log n/\log n)}$.

Applications

Despite the large (exponential in n) approximation factor, the LLL algorithm has found numerous applications and lead to the solution of many algorithmic problems in computer science. The number and variety of applications is too large to give a comprehensive list. Some of the most representative applications in different areas of computer science are mentioned below.

The first motivating applications of lattice basis reduction were the solution of integer programs with a fixed number of variables and the factorization of polynomials with rationals coefficients. (See [11] [8], and [5, chap. 16].) Other classic applications are the solution of random instances of low-density subset-sum problems, breaking (truncated) linear congruential pseudorandom generators, simultaneous Diophantine approximation, and the disproof of Mertens' conjecture. (See [8] and [5, chap. 17].)

More recently, lattice basis reduction has been extensively used to solve many problems in cryptanalysis and coding theory, including breaking several variants of the RSA cryptosystem and the DSA digital signature algorithm, finding small solutions to modular equations, and list decoding of CRT (Chinese Reminder Theorem) codes. The reader is referred to [7,13] for a survey of recent applications, mostly in the area of cryptanalysis.

One last class of applications of lattice problems is the design of cryptographic functions (e. g., collision resistant hash functions, public key encryption schemes, etc.) based on the apparent intractability of solving SVP_γ within small approximation factors. The reader is referred to [12, chap. 8] and [13] for a survey of such applications, and further pointers to relevant literature. One distinguishing feature of many such lattice based cryptographic functions is that they can be proved to be hard to break *on the average*, based on a *worst-case* intractability assumption about the underlying lattice problem.

Open Problems

The main open problems in the computational study of lattices is to determine the complexity of approximate SVP_γ and CVP_γ for approximation factors $\gamma = n^c$ polynomial in the rank of the lattice. Specifically,

- Are there polynomial time algorithm that solve SVP_γ or CVP_γ for polynomial factors $\gamma = n^c$? (Finding such algorithms even for very large exponent c would be a major breakthrough in computer science.)
- Is there an $\epsilon > 0$ such that approximating SVP_γ or CVP_γ to within $\gamma = n^\epsilon$ is NP-hard? (The strongest known inapproximability results [4] are for factors of the form $n^{O(1/\log\log n)}$ which grow faster than any polylogarithmic function, but slower than any polynomial.)

There is theoretical evidence that for large polynomials factors $\gamma = n^c$, SVP_γ and CVP_γ are not NP-hard. Specifically, both problems belong to complexity class coAM for approximation factor $\gamma = O(\sqrt{n/\log n})$. (See [12, chap. 9].) So, the problems cannot be NP-hard within such factors unless the polynomial hierarchy PH collapses.

URL to Code

The LLL lattice reduction algorithm is implemented in most library and packages for computational algebra, e. g.,

- GAP (http://www.gap-system.org)
- LiDIA (http://www.cdc.informatik.tu-darmstadt.de/TI/LiDIA/)
- Magma (http://magma.maths.usyd.edu.au/magma/)
- Maple (http://www.maplesoft.com/)
- Mathematica (http://www.wolfram.com/products/mathematica/index.html)
- NTL (http://shoup.net/ntl/).

NTL also includes an implementation of Block Korkine-Zolotarev reduction that has been extensively used for cryptanalysis applications.

Cross References

▶ Cryptographic Hardness of Learning
▶ Knapsack
▶ Learning Heavy Fourier Coefficients of Boolean Functions
▶ Quantum Algorithm for the Discrete Logarithm Problem
▶ Quantum Algorithm for Factoring
▶ Sphere Packing Problem

Recommended Reading

1. Ajtai, M., Kumar, R., Sivakumar, D.: A sieve algorithm for the shortest lattice vector problem. In: Proceedings of the thirty-third annual ACM symposium on theory of computing – STOC 2001, Heraklion, Crete, Greece, July 2001, pp 266–275. ACM, New York (2001)
2. Babai, L.: On Lovasz' lattice reduction and the nearest lattice point problem. Combinatorica **6**(1), 1–13 (1986). Preliminary version in STACS 1985
3. Cassels, J.W.S.: An introduction to the geometry of numbers. Springer, New York (1971)
4. Dinur, I., Kindler, G., Raz, R., Safra, S.: Approximating CVP to within almost-polynomial factors is NP-hard. Combinatorica **23**(2), 205–243 (2003). Preliminary version in FOCS 1998
5. von zur Gathen, J., Gerhard, J.: Modern Comptuer Algebra, 2nd edn. Cambridge (2003)
6. Grotschel, M., Lovász, L., Schrijver, A.: Geometric algorithms and combinatorial optimization. Algorithms and Combinatorics, vol. 2, 2nd edn. Springer (1993)
7. Joux, A., Stern, J.: Lattice reduction: A toolbox for the cryptanalyst. J. Cryptolo. **11**(3), 161–185 (1998)
8. Kannan, R.: Annual reviews of computer science, vol. 2, chap. "Algorithmic geometry of numbers", pp. 231–267. Annual Review Inc., Palo Alto, California (1987)
9. Kannan, R.: Minkowski's convex body theorem and integer programming. Math. Oper. Res. **12**(3), 415–440 (1987)
10. Khot, S.: Hardness of Approximating the Shortest Vector Problem in Lattices. J. ACM **52**(5), 789–808 (2005). Preliminary version in FOCS 2004
11. Lenstra, A.K., Lenstra, Jr., H.W., Lovász, L.: Factoring polynomials with rational coefficients. Math Ann. **261**, 513–534 (1982)
12. Micciancio, D., Goldwasser, S.: Complexity of Lattice Problems: A Cryptographic Perspective. The Kluwer International Series in Engineering and Computer Science, vol. 671. Kluwer Academic Publishers, Boston, Massachusetts (2002)
13. Nguyen, P., Stern, J.: The two faces of lattices in cryptology. In: J. Silverman (ed.) Cryptography and lattices conference – CaLC 2001, Providence, RI, USA, March 2001. Lecture Notes in Computer Science, vol. 2146, pp. 146–180. Springer, Berlin (2001)
14. Schnorr, C.P.: Fast LLL-type lattice reduction. Inform. Comput. **204**(1), 1–25 (2006)
15. Vazirani, V.V.: Approximation Algorithms. Springer (2001)

Similarity between Compressed Strings

2005; Kim, Amir, Landau, Park

Jin Wook Kim[1], Amihood Amir[2], Gad M. Landau[3], Kunsoo Park[4]
[1] HM Research, Seoul, Korea
[2] Department of Computer Science, Bar-Ilan University, Ramat-Gan, Israel
[3] Department of Computer Science, University of Haifa, Haifa, Israel
[4] School of Computer Science and Engineering, Seoul National University, Seoul, Korea

Keywords and Synonyms

Similarity between compressed strings; Compressed approximate string matching; Alignment between compressed strings

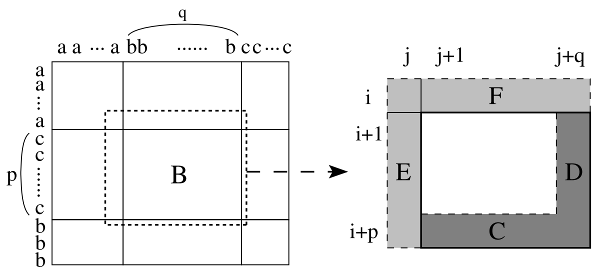
Problem Definition

The problem of computing similarity between two strings is concerned with comparing two strings using some scoring metric. There exist various scoring metrics and a popular one is the Levenshtein distance (or edit distance) metric. The standard solution for the Levenshtein distance metric was proposed by Wagner and Fischer [13], which is based on dynamic programming. Other widely used scoring metrics are the longest common subsequence metric, the weighted edit distance metric, and the affine gap penalty metric. The affine gap penalty metric is the most general, and it is a quite complicated metric to deal with. Table 1 shows the differences between the four metrics.

The problem considered in this entry is the similarity between two compressed strings. This problem is concerned with efficiently computing similarity without decompressing two strings. The compressions used for this

Similarity between Compressed Strings, Table 1
Various scoring metrics

Metric	Match	Mismatch	Indel	Indel of k characters
Longest common subsequence	1	0	0	0
Levenshtein distance	0	1	1	k
Weighted edit distance	0	δ	μ	$k\mu$
Affine gap penalty	1	$-\delta$	$-\gamma-\mu$	$-\gamma-k\mu$



Similarity between Compressed Strings, Figure 1
Dynamic programming table for strings $a^r c^p b^t$ and $a^s b^q c^u$ is divided into 9 blocks. For one of the blocks, e. g., *B*, only the bottom row *C* and the rightmost column *D* are computed from *E* and *F*

problem in the literature are run-length encoding and Lempel-Ziv (LZ) compression [14].

Run-Length Encoding

A string S is run-length encoded if it is described as an ordered sequence of pairs (σ, i), often denoted "σ^i", each consisting of an alphabet symbol, σ, and an integer, i. Each pair corresponds to a *run* in S, consisting of i consecutive occurrences of σ. For example, the string *aaabbbbaccccbb* can be encoded $a^3b^4a^1c^4b^2$ or, equivalently, $(a,3)(b,4)(a,1)(c,4)(b,2)$. Let A and B be two strings with lengths n and m, respectively. Let A' and B' be the run-length encoded strings of A and B, and n' and m' be the lengths of A' and B', respectively.

Problem 1
INPUT: *Two run-length encoded strings A' and B', a scoring metric d.*
OUTPUT: *The similarity between A' and B' using d.*

LZ Compression

Let X and Y be two strings with length $O(n)$. Let X' and Y' be the LZ compressed strings of X and Y, respectively. Then the lengths of X' and Y' are $O(hn/\log n)$, where $h \le 1$ is the entropy of strings X and Y.

Problem 2
INPUT: *Two LZ compressed strings X' and Y', a scoring metric d.*
OUTPUT: *The similarity between X' and Y' using d.*

Block Computation

To compute similarity between compressed strings efficiently, one can use a block computation method. Dynamic programming tables are divided into submatrices, which are called "*blocks*". For run-length encoded strings, a block is a submatrix made up of two runs – one of A and one of B. For LZ compressed strings, a block is a submatrix made up of two phrases – one phrase from each string. See [5] for more details. Then, blocks are computed from left to right and from top to bottom. For each block, only the bottom row and the rightmost column are computed. Figure 1 shows an example of block computation.

Key Results

The problem of computing similarity of two run-length encoded strings, A' and B', has been studied for various scoring metrics. Bunke and Csirik [4] presented the first solution to Problem 1 using the longest common subsequence metric. The algorithm is based on block computation of the dynamic programming table.

Theorem 1 (Bunke and Csirik 1995 [4]) *A longest common subsequence of run-length encoded strings A' and B' can be computed in $O(nm' + n'm)$ time.*

For the Levenshtein distance metric, Arbell, Landau, and Mitchell [2] and Mäkinen, Navarro, and Ukkonen [10] presented $O(nm' + n'm)$ time algorithms, independently. These algorithms are extensions of the algorithm of Bunke and Csirik.

Theorem 2 (Arbell, Landau, and Mitchell 2002 [2], Mäkinen, Navarro, and Ukkonen [10]) *The Levenshtein distance between run-length encoded strings A' and B' can be computed in $O(nm' + n'm)$ time.*

For the weighted edit distance metric, Crochemore, Landau, and Ziv-Ukelson [6] and Mäkinen, Navarro, and Ukkonen [11] gave $O(nm' + n'm)$ time algorithms using techniques completely different from each other. The algorithm of Crochemore, Landau, and Ziv-Ukelson [6] is based on the technique which is used in the LZ compressed pattern matching algorithm [6], and the algorithm of Mäkinen, Navarro, and Ukkonen [11] is an extension of the algorithm for the Levenshtein distance metric.

Theorem 3 (Crochemore, Landau, and Ziv-Ukelson 2003 [6] Mäkinen, Navarro, and Ukkonen [11]) *The weighted edit distance between run-length encoded strings A' and B' can be computed in $O(nm' + n'm)$ time.*

For the affine gap penalty metric, Kim, Amir, Landau, and Park [8] gave an $O(nm' + n'm)$ time algorithm. To compute similarity in this metric efficiently, the problem is converted into a path problem on a directed acyclic graph and some properties of maximum paths in this graph are used. It is not necessary to build the graph explicitly since they came up with new recurrences using the properties of the graph.

Theorem 4 (Kim, Amir, Landau, and Park 2005 [8]) *The similarity between run-length encoded strings A' and B' in the affine gap penalty metric can be computed in $O(nm' + n'm)$ time.*

The above results show that comparison of run-length encoded strings using the longest common subsequence metric is successfully extended to more general scoring metrics.

For the longest common subsequence metric, there exist improved algorithms. Apostolico, Landau, and Skiena [1] gave an $O(n'm' \log(n'm'))$ time algorithm. This algorithm is based on tracing specific optimal paths.

Theorem 5 (Apostolico, Landau, and Skiena 1999 [1]) *A longest common subsequence of run-length encoded strings A' and B' can be computed in $O(n'm' \log(n' + m'))$ time.*

Mitchell [12] obtained an $O((d + n' + m') \log(d + n' + m'))$ time algorithm, where d is the number of matches of compressed characters. This algorithm is based on computing geometric shortest paths using special convex distance functions.

Theorem 6 (Mitchell 1997 [12]) *A longest common subsequence of run-length encoded strings A' and B' can be computed in $O((d + n' + m') \log(d + n' + m'))$ time, where d is the number of matches of compressed characters.*

Mäkinen, Navarro, and Ukkonen [11] conjectured an $O(n'm')$ time algorithm on average under the assumption that the lengths of the runs are equally distributed in both strings.

Conjecture 1 (Mäkinen, Navarro, and Ukkonen 2003 [11]) *A longest common subsequence of run-length encoded strings A' and B' can be computed in $O(n'm')$ time on average.*

For Problem 2, Crochemore, Landau, and Ziv-Ukelson [6] presented a solution using the additive gap penalty metric. The additive gap penalty metric consists of 1 for match, $-\delta$ for mismatch, and $-\mu$ for indel, which is almost the same as the weighted edit distance metric.

Theorem 7 (Crochemore, Landau, and Ziv-Ukelson 1993 [6]) *The similarity between LZ compressed strings X' and Y' in the additive gap penalty metric can be computed in $O(hn^2/\log n)$ time, where $h \leq 1$ is the entropy of strings X and Y.*

Applications

Run-length encoding serves as a popular image compression technique, since many classes of images (e. g., binary images in facsimile transmission or for use in optical character recognition) typically contain large patches of identically-valued pixels. Approximate matching on images can be a useful tool to handle distortions. Even a one-dimensional compressed approximate matching algorithm would be useful to speed up two-dimensional approximate matching allowing mismatches and even rotations [3,7,9].

Open Problems

The worst-case complexity of the problem is not fully understood. For the longest common subsequence metric, there exist some results whose time complexities are better than $O(nm' + n'm)$ to compute the similarity of two run-length encoded strings [1,11,12]. It remains open to extend these results to the Levenshtein distance metric, the weighted edit distance metric and the affine gap penalty metric.

In addition, for the longest common subsequence metric, it is an open problem to prove Conjecture 1.

Cross References

► Compressed Pattern Matching
► Local Alignment (with Affine Gap Weights)
► Sequential Approximate String Matching

Recommended Reading

1. Apostolico, A., Landau, G.M., Skiena, S.: Matching for Run Length Encoded Strings. J. Complex. **15**(1), 4–16 (1999)
2. Arbell, O., Landau, G.M., Mitchell, J.: Edit Distance of Run-Length Encoded Strings. Inf. Proc. Lett. **83**(6), 307–314 (2002)
3. Baeza-Yates, R., Navaro, G.: New Models and Algorithms for Multidimensional Approximate Pattern Matching. J. Discret. Algorithms **1**(1), 21–49 (2000)
4. Bunke, H., Csirik, H.: An Improved Algorithm for Computing the Edit Distance of Run Length Coded Strings. Inf. Proc. Lett. **54**, 93–96 (1995)
5. Crochemore, M., Landau, G.M., Schieber, B., Ziv-Ukelson, M.: Re-Use Dynamic Programming for Sequence Alignment: An Algorithmic Toolkit. In: Iliopoulos, C.S., Lecroq, T. (eds.) String Algorithmics, pp. 19–59. King's College London Publications, London (2005)
6. Crochemore, M., Landau, G.M., Ziv-Ukelson, M.: A Subquadratic Sequence Alignment Algorithm for Unrestricted Scoring Matrices. SIAM J. Comput. **32**(6), 1654–1673 (2003)
7. Fredriksson, K., Navarro, G., Ukkonen, E.: Sequential and Indexed Two-Dimensional Combinatorial Template Matching Allowing Rotations. Theor. Comput. Sci. **347**(1–2), 239–275 (2005)
8. Kim, J.W., Amir, A., Landau, G.M., Park, K.: Computing Similarity of Run-Length Encoded Strings with Affine Gap Penalty. In: Proc. 12th Symposium on String Processing and Information Retrieval (SPIRE'05). LNCS, vol. 3772, pp. 440–449 (2005)

9. Krithivasan, K., Sitalakshmi, R.: Efficient Two-Dimensional Pattern Matching in The Presence of Errors. Inf. Sci. **43**, 169–184 (1987)
10. Mäkinen, V., Navarro, G., Ukkonen, E.: Approximate Matching of Run-Length Compressed Strings. In: Proc. 12th Symposium on Combinatorial Pattern Matching (CPM'01). LNCS, vol. 2089, pp. 31–49 (2001)
11. Mäkinen, V., Navarro, G., Ukkonen, E.: Approximate Matching of Run-Length Compressed Strings. Algorithmica **35**, 347–369 (2003)
12. Mitchell, J.: A Geometric Shortest Path Problem, with Application to Computing a Longest Common Subsequence in Run-Length Encoded Strings. Technical Report, Dept. of Applied Mathematics, SUNY Stony Brook (1997)
13. Wagner, R.A., Fischer, M.J.: The String-to-String correction Problem. J. ACM **21**(1), 168–173 (1974)
14. Ziv, J., Lempel, A.: Compression of Individual Sequences via Variable Rate Coding. IEEE Trans. Inf. Theory **24**(5), 530–536 (1978)

Single-Source Fully Dynamic Reachability

2005; Demetrescu, Italiano

CAMIL DEMETRESCU, GIUSEPPE F. ITALIANO
Department of Computer & Systems Science,
University of Rome, Rome, Italy

Keywords and Synonyms

Single-source fully dynamic transitive closure

Problem Definition

A dynamic graph algorithm maintains a given property $\mathcal{P}$ on a graph subject to dynamic changes, such as edge insertions, edge deletions and edge weight updates. A dynamic graph algorithm should process queries on property $\mathcal{P}$ quickly, and perform update operations faster than recomputing from scratch, as carried out by the fastest static algorithm. An algorithm is *fully dynamic* if it can handle both edge insertions and edge deletions and *partially dynamic* if it can handle either edge insertions or edge deletions, but not both.

Given a graph with n vertices and m edges, the *transitive closure* (or *reachability*) problem consists of building an $n \times n$ Boolean matrix M such that $M[x, y] = 1$ if and only if there is a directed path from vertex x to vertex y in the graph. The fully dynamic version of this problem can be defifined as follows:

Definition 1 (Fully dynamic reachability problem) The *fully dynamic reachability problem* consists of maintaining a directed graph under an intermixed sequence of the following operations:

- `insert`(u,v): insert edge (u,v) into the graph.
- `delete`(u,v): delete edge (u,v) from the graph.
- `reachable`(x,y): return *true* if there is a directed path from vertex x to vertex y, and *false* otherwise.

This entry addresses the *single-source* version of the fully-dynamic reachability problem, where one is only interested in queries with a fixed source vertex s. The problem is defined as follows:

Definition 2 (Single-source fully dynamic reachability problem) The *fully dynamic single-source reachability problem* consists of maintaining a directed graph under an intermixed sequence of the following operations:

- `insert`(u,v): insert edge (u,v) into the graph.
- `delete`(u,v): delete edge (u,v) from the graph.
- `reachable`(y): return *true* if there is a directed path from the source vertex s to vertex y, and *false* otherwise.

Approaches

A simple-minded solution to the problem of Definition would be to keep explicit reachability information from the source to all other vertices and update it by running any graph traversal algorithm from the source s after each insert or delete. This takes $O(m + n)$ time per operation, and then reachability queries can be answered in constant time.

Another simple-minded solution would be to answer queries by running a point-to-point reachability computation, without the need to keep explicit reachability information up to date after each insertion or deletion. This can be done in $O(m + n)$ time using any graph traversal algorithm. With this approach, queries are answered in $O(m + n)$ time and updates require constant time. Notice that the time required by the slowest operation is $O(m+n)$ for both approaches, which can be as high as $O(n^2)$ in the case of dense graphs.

The first improvement upon these two basic solutions is due to Demetrescu and Italiano, who showed how to support update operations in $O(n^{1.575})$ time and reachability queries in $O(1)$ time [1] in a directed acyclic graph. The result is based on a simple reduction of the single-source problem of Definition to the all-pairs problem of Definition. Using a result by Sankowski [2], the bounds above can be extended to the case of general directed graphs.

Key Results

This Section presents a simple reduction presented in [1] that allows it to keep explicit single-source reachability information up to date in subquadratic time per operation in a directed graph subject to an intermixed sequence of edge insertions and edge deletions. The bounds reported in this entry were originally presented for the case of directed acyclic graphs, but can be extended to general directed graphs using the following theorem from [2]:

Theorem 1 *Given a general directed graph with n vertices, there is a data structure for the fully dynamic reachability problem that supports each insertion/deletion in* $O(n^{1.575})$ *time and each reachability query in* $O(n^{0.575})$ *time. The algorithm is randomized with one-sided error.*

The idea described in [1] is to maintain reachability information from the source vertex s to all other vertices explicitly by keeping a Boolean array R of size n such that $R[y] = 1$ if and only if there is a directed path from s to y. An instance D of the data structure for fully dynamic reachability of Theorem is also maintained. After each insertion or deletion, it is possible to update D in $O(n^{1.575})$ time and then rebuild R in $O(n \cdot n^{0.575}) = O(n^{1.575})$ time by letting $R[y] \leftarrow D$.`reachable` (s,y) for each vertex y. This yields the following bounds for the single-source fully dynamic reachability problem:

Theorem 2 *Given a general directed graph with n vertices, there is a data structure for the single-source fully dynamic reachability problem that supports each insertion/deletion in* $O(n^{1.575})$ *time and each reachability query in* $O(1)$ *time.*

Applications

The graph reachability problem is particularly relevant to the field of databases for supporting transitivity queries on dynamic graphs of relations [3]. The problem also arises in many other areas such as compilers, interactive verification systems, garbage collection, and industrial robotics.

Open Problems

An important open problem is whether one can extend the result described in this entry to maintain fully dynamic single-source shortest paths in subquadratic time per operation.

Cross References

► Trade-Offs for Dynamic Graph Problems

Recommended Reading

1. Demetrescu, C., Italiano, G.: Trade-offs for fully dynamic reachability on dags: Breaking through the $O(n^2)$ barrier. J. Assoc. Comput. Machin. (JACM) **52**, 147–156 (2005)
2. Sankowski, P.: Dynamic transitive closure via dynamic matrix inverse. In: FOCS '04: Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS'04), pp. 509–517. IEEE Computer Society, Washington DC (2004)
3. Yannakakis, M.: Graph-theoretic methods in database theory. In: Proc. 9-th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Nashville, 1990 pp. 230–242

Single-Source Shortest Paths

1999; Thorup

Seth Pettie
Department of Computer Science,
University of Michigan, Ann Arbor, MI, USA

Keywords and Synonyms

Shortest route; Quickest route

Problem Definition

The *single source* shortest path problem (SSSP) is, given a graph $G = (V, E, \ell)$ and a *source* vertex $s \in V$, to find the shortest path from s to every $v \in V$. The difficulty of the problem depends on whether the graph is directed or undirected and the assumptions placed on the length function ℓ. In the most general situation $\ell\colon E \to \mathbb{R}$ assigns arbitrary (positive & negative) real lengths. The algorithms of Bellman-Ford and Edmonds [1,4] may be applied in this situation and have running times of roughly $O(mn)$,[1] where $m = |E|$ and $n = |V|$ are the number of edges and vertices. If ℓ assigns only *non-negative* real edge lengths then the algorithms of Dijkstra and Pettie-Ramachandran [4,14] may be applied on directed and undirected graphs, respectively. These algorithms include a *sorting bottleneck* and, in the worst case, take $\Omega(m + n \log n)$ time.[2]

A common assumption is that ℓ assigns *integer* edge lengths in the range $\{0, \ldots, 2^w - 1\}$ or $\{-2^{w-1}, \ldots, 2^{w-1} - 1\}$ and that the machine is a w-bit *word RAM*; that is, each edge length fits in one register. For general integer edge lengths the best SSSP algorithms improve on Bellman-Ford and Edmonds by a factor of roughly $\sqrt{n}$ [7]. For non-negative integer edge lengths the best SSSP algorithms are faster than Dijkstra and Pettie-Ramachandran

[1]Edmonds's algorithm works for undirected graphs and presumes that there are no negative length simple cycles.

[2]The [14] algorithm actually runs in $O(m + n \log\log n)$ time if the ratio of any two edge lengths is polynomial in n.

by up to a logarithmic factor. They are frequently based on integer priority queues [10].

Key Results

Thorup's primary result [17] is an optimal linear time SSSP algorithm for undirected graphs with integer edge lengths. This is the first and only linear time shortest path algorithm that does not make serious assumptions on the class of input graphs.

Theorem 1 *There is a SSSP algorithm for integer-weighted undirected graphs that runs in $O(m)$ time.*

Thorup avoids the sorting bottleneck inherent in Dijkstra's algorithm by precomputing (in linear time) a *component hierarchy*. The algorithm of [17] operates in a manner similar to Dijkstra's algorithm [4] but uses the component hierarchy to identify groups of vertices that can be visited in any order. In later work, Thorup [18] extended this approach to work when the edge lengths are floating-point numbers.[3]

Thorup's hierarchy-based approach has since been extended to directed and/or real-weighted graphs, and to solve the *all pairs* shortest path (APSP) problem [12,13,14]. The generalizations to related SSSP problems are summarized by below. See [12,13] for hierarchy-based APSP algorithms.

Theorem 2 *(Hagerup [9], 2000) A component hierarchy for a directed graph $G = (V, E, \ell)$, where $\ell\colon E \to \{0, \ldots, 2^w - 1\}$, can be constructed in $O(m \log w)$ time. Thereafter SSSP from any source can be computed in $O(m + n \log\log n)$ time.*

Theorem 3 *(Pettie and Ramachandran [14], 2005) A component hierarchy for an undirected graph $G = (V, E, \ell)$, where $\ell\colon E \to \mathbb{R}^+$, can be constructed in $O(m\alpha(m, n) + \min\{n \log\log r, n \log n\})$ time, where r is the ratio of the maximum-to-minimum edge length. Thereafter SSSP from any source can be computed in $O(m \log \alpha(m, n))$ time.*

The algorithms of Hagerup [9] and Pettie-Ramachandran [14] take the same basic approach as Thorup's algorithm: use some kind of component hierarchy to identify groups of vertices that can safely be visited in any order. However, the assumption of directed graphs [9] and real edge lengths [14] renders Thorup's hierarchy inapplicable or inefficient. Hagerup's component hierarchy is based on a directed analogue of the minimum spanning tree. The Pettie-Ramachandran algorithm enforces a certain degree of balance in its component hierarchy and, when computing SSSP, uses a specialized priority queue to take advantage of this balance.

[3]There is some flexibility in the definition of *shortest path* since floating-point addition is neither commutative nor associative.

Applications

Shortest path algorithms are frequently used as a subroutine in other optimization problems, such as flow and matching problems [1] and facility location [19]. A widely used commercial application of shortest path algorithms is finding efficient routes on road networks, e. g., as provided by Google Maps, MapQuest, or Yahoo Maps.

Open Problems

Thorup's SSSP algorithm [17] runs in linear time and is therefore optimal. The main open problem is to find a linear time SSSP algorithm that works on *real*-weighted *directed* graphs. For real-weighted undirected graphs the best running time is given in Theorem 3. For integer-weighted directed graphs the fastest algorithms are based on Dijkstra's algorithm (not Theorem 2) and run in $O(m\sqrt{\log\log n})$ time (randomized) and deterministically in $O(m + n \log\log n)$ time.

Problem 1 *Is there an $O(m)$ time SSSP algorithm for integer-weighted directed graphs?*

Problem 2 *Is there an $O(m) + o(n \log n)$ time SSSP algorithm for real-weighted graphs, either directed or undirected?*

The complexity of SSSP on graphs with positive & negative edge lengths is also open.

Experimental Results

Asano and Imai [2] and Pettie et al. [15] evaluated the performance of the hierarchy-based SSSP algorithms [14,17]. There have been a number of studies of SSSP algorithms on integer-weighted directed graphs; see [8] for the latest and references to many others. The trend in recent years is to find practical preprocessing schemes that allow for very quick point-to-point shortest path queries. See [3,11,16] for recent work in this area.

Data Sets

See [5] for a number of US and European road networks.

URL to Code

See [6] and [5].

Cross References

► All Pairs Shortest Paths via Matrix Multiplication

Recommended Reading

1. Ahuja, R.K., Magnati, T.L., Orlin, J.B.: Network Flows: Theory, Algorithms, and Applications. Prentice Hall, Englewood Cliffs (1993)
2. Asano, Y., Imai, H.: Practical efficiency of the linear-time algorithm for the single source shortest path problem. J. Oper. Res. Soc. Jpn. **43**(4), 431–447 (2000)
3. Bast, H., Funke, S., Matijevic, D., Sanders, P., Schultes, D.: In transit to constant shortest-path queries in road networks. In: Proceedings 9th Workshop on Algorithm Engineering and Experiments (ALENEX), 2007
4. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms. MIT Press, Cambridge (2001)
5. Demetrescu, C., Goldberg, A.V., Johnson, D.: 9th DIMACS Implementation Challege—Shortest Paths. http://www.dis.uniroma1.it/~challenge9/ (2006)
6. Goldberg, A.V.: AVG Lab. http://www.avglab.com/andrew/
7. Goldberg, A.V.: Scaling algorithms for the shortest paths problem. SIAM J. Comput. **24**(3), 494–504 (1995)
8. Goldberg, A.V.: Shortest path algorithms: Engineering aspects. In: Proc. 12th Int'l Symp. on Algorithms and Computation (ISAAC). LNCS, vol. 2223, pp. 502–513. Springer, Berlin (2001)
9. Hagerup, T.: Improved shortest paths on the word RAM. In: Proc. 27th Int'l Colloq. on Automata, Languages, and Programming (ICALP). LNCS vol. 1853, pp. 61–72. Springer, Berlin (2000)
10. Han, Y., Thorup, M.: Integer sorting in $O(n\sqrt{\log\log n})$ expected time and linear space. In: Proc. 43rd Symp. on Foundations of Computer Science (FOCS), 2002, pp. 135–144
11. Knopp, S., Sanders, P., Schultes, D., Schulz, F., Wagner, D.: Computing many-to-many shortest paths using highway hierarchies. In: Proceedings 9th Workshop on Algorithm Engineering and Experiments (ALENEX), 2007
12. Pettie, S.: On the comparison-addition complexity of all-pairs shortest paths. In: Proc. 13th Int'l Symp. on Algorithms and Computation (ISAAC), 2002, pp. 32–43
13. Pettie, S.: A new approach to all-pairs shortest paths on real-weighted graphs. Theor. Comput. Sci. **312**(1), 47–74 (2004)
14. Pettie, S., Ramachandran, V.: A shortest path algorithm for real-weighted undirected graphs. SIAM J. Comput. **34**(6), 1398–1431 (2005)
15. Pettie, S., Ramachandran, V., Sridhar, S.: Experimental evaluation of a new shortest path algorithm. In: Proc. 4th Workshop on Algorithm Engineering and Experiments (ALENEX), 2002, pp. 126–142
16. Sanders, P., Schultes, D.: Engineering Highway Hierarchies. In: Proc. 14th European Symposium on Algorithms (ESA), 2006, pp. 804–816
17. Thorup, M.: Undirected single-source shortest paths with positive integer weights in linear time. J. ACM **46**(3), 362–394 (1999)
18. Thorup, M.: Floats, integers, and single source shortest paths. J. Algorithms **35** (2000)
19. Thorup, M.: Quick and good facility location. In: Proceedings 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 2003, pp. 178–185

Ski Rental Problem

1990; Karlin, Manasse, McGeogh, Owicki

MARK S. MANASSE
Microsoft Research, Mountain View, CA, USA

Index Terms

Ski-rental problem, Competitive algorithms, Deterministic and randomized algorithms, On-line algorithms

Keywords and Synonyms

Oblivious adversaries, Worst-case approximation, Metrical task systems

Problem Definition

The ski rental problem was developed as a pedagogical tool for understanding the basic concepts in some early results in on-line algorithms.[1] The ski rental problem considers the plight of one consumer who, in order to socialize with peers, is forced to engage in a variety of athletic activities, such as skiing, bicycling, windsurfing, rollerblading, sky diving, scuba-diving, tennis, soccer, and ultimate Frisbee, each of which has a set of associated apparatus, clothing, or protective gear.

In all of these, it is possible either to purchase the accoutrements needed, or to rent them. For the purpose of this problem, it is assumed that one-time rental is less expensive than purchasing. It is also assumed that purchased items are durable, and suitable for reuse for future activities of the same type without further expense, until the items wear out (which occurs at the same rate for all users), are outgrown, become unfashionable, or are disposed of

[1] In the interest of full disclosure, the earliest presentations of these results described the problem as the wedding-tuxedo-rental problem. Objections were presented that this was a gender-biased name for the problem, since while groomsmen can rent their wedding apparel, bridesmaids usually cannot. A further complication, owing to the difficulty of instantaneously producing fitted garments or ski equipment outlined below, suggests that some complications could have been avoided by focusing on the dilemma of choosing between daily lift passes or season passes, although this leads to the pricing complexities of purchasing season passes well in advance of the season, as opposed to the higher cost of purchasing them at the mountain during the ski season. A similar problem could be derived from the question as to whether to purchase the daily newspaper at a newsstand or to take a subscription, after adding the challenge that one's peers will treat one contemptuously if one has not read the news on days on which they have.

to make room for other purchased items. The social consumer must make the decision to rent or buy for each event, although it is assumed that the consumer is sufficiently parsimonious as to abjure rental if already in possession of serviceable purchased equipment. Whether purchases are as easy to arrange as rentals, or whether some advance planning is required (to mount bindings on a ski, say) is a further detail considered in this problem. It is assumed that the social consumer has no particular independent interest in these activities, and engages in these activities only to socialize with peers who choose to engage in these activities disregarding the consumer's desires.

These putative peers are more interested in demonstrating the superiority of their financial acumen to that of the social consumer in question than they are in any particular activity. To that end, the social consumer is taunted mercilessly based on the ratio of his/her total expenses on rentals and purchases to theirs. Consequently, the peers endeavor to invite the social consumer to engage in events while they are costly to him/her, and once the activities are free to the social consumer, if continued activity would be costly to them, cease. But, to present an illusion of fairness, skis, both rented and purchased, have the same cost for the peers as they do for the social consumer in question. The ski rental problem takes a very restricted setting. It assumes that purchased ski equipment never needs replacement, and that there are no costs to a ski trip other than the skis (thus, no cost for the gasoline, for the lift and/or speeding tickets, for the hot chocolates during skiing, or for the après-ski liqueurs and meals). It is assumed that the social consumer experiences no physical disabilities preventing him/her from skiing, and has no impending restrictions to his/her participation in ski trips (obviously, a near-term-fatal illness or an anticipated conviction leading to confinement for life in a penitentiary would eliminate any potential interest in purchasing alpine equipment—when the ratio of purchase to rental exceeds the maximum need for equipment, one should always rent). It is assumed that the social consumer's peers have disavowed any interest in activities other than skiing, and that the closet, basement, attic, garage, or storage locker included in the social consumer's rent or mortgage (or necessitated by other storage needs) has sufficient capacity to hold purchased ski equipment without entailing the disposal of any potentially useful items. Bringing these complexities into consideration brings one closer to the hardware-based problems which initially inspired this work.

The impact of invitations issued with sufficient time allowed for purchasing skis, as well as those without, will be considered.

Given all of that, what ratio of expenses can the social consumer hope to attain? What ratio can the social consumer not expect to beat? These are the basic questions of competitive analysis.

The impact of keeping secrets from one's peers is further considered. Rather than a fixed strategy for when to purchase skis, the social consumer may introduce an element of chance into the process. If the peers are able to observe his/her ski equipment and notice when it changes from rented skis to purchased skis, and change their schedule for alpine recreation in light of this observation, randomness provides no advantages. If, on the other hand, the social consumer announces to the peers, in advance of the first trip, how he/she will decide when the time is right for purchasing skis, including any use of probabilistic techniques, and they then decide on the schedule for ski trips for the coming winter, a deterministic decision procedure generally produces a larger competitive ratio than does a randomized procedure.

Key Results

Given an unbounded sequence of skiing trips, one should eventually purchase skis if the cost of renting skis, r, is positive. In particular, let the cost of purchasing skis be some number $p \geq r$. If one never intends to make a purchase, one's cost for the season will be rn, where n is the number of ski trips in which one participates. If n exceeds p/r, one's cost will exceed the price of purchasing skis; as n continues to increase, the ratio of one's costs to those of one's peers increases to nr/p, which grows unboundedly with n, since your peers, knowing that n exceeds p/r, will have purchased skis prior to the first trip.

On the other hand, if one rushes out to purchase skis upon being told that the ski season is approaching, one's peers will decide that this season looks inopportune, and that skiing is passé, leaving their costs at zero, and one's costs at p, leaving an infinite ratio between one's costs and theirs; if one chooses to defer the purchase until after one's first ski trip, this produces the less unfavorable ratio p/r or $1 + p/r$, depending on whether the invitation left one time to purchase skis before the first trip or not.

Suppose one chooses, instead, to defer one's purchase until after one has made k rentals, but before ski trip $k + 1$. One's costs are then bounded by $kr + p$. After k ski trips, the cost to one's peers will be the lesser of kr and p (as one's peers will have decided whether to rent or buy for the season upon knowing one's plans, which in this case amounts to knowing k), for a ratio equal to the larger of $1 + kr/p$ and $1 + p/kr$. Were they to choose to terminate the activity earlier (so $n < k$), the ratio would be only the greater of kr/p and 1, which is guaranteed to be less than the sum of the two—one's peers would be shirking their opportunity to make one's behavior look foolish were they to allow one to stop skiing prior to one's purchase of a pair of skis!

It is certain, since kr/p and p/kr are reciprocals, that one of them is at least equal to 1, ensuring that one will be compelled to spend at least twice as much as one's peers.

The analysis above applies to the case where ski trips are announced without enough warning to leave one time to buy skis. Purchases in that case are not instantaneous; in contrast, if one is able to purchase skis on demand, the cost to one's peers changes to the lesser of $(k+1)\,r$ and p. The overall results are not much different; the ratio choices become the larger of $1 + kr/p$ and $1 + \left(p - r\right)/((k+1)\,r)$.

When probabilistic algorithms are considered with oblivious frenemies (those who know the way in which random choices will affect one's purchasing decisions, but who do not take time to notice that one's skis are no longer marked with the name and phone number of a rental agency), one can appear more thrifty.

A randomized algorithm can be viewed as a distribution over deterministic algorithms. No good algorithm can purchase skis prior to the first invitation, lest it exhibit infinite regrettability (some positive cost compared to zero). A good algorithm must purchase skis by the time one's peers will have, otherwise one's cost ratio continues to increase with the number of ski trips. Moreover, the ratio should be the same after every ski trip; if not, then there is an earliest ratio not equal to the largest, and probabilities can be adjusted to change this earliest ratio to be closer to the largest while decreasing all larger ratios.

Consider, for example, the case of $p = 2r$, with purchases allowed at the time of an invitation. The best deterministic ratio in this case is 1.5. It is only necessary to choose a probability q, the probability of purchasing at the time of the first invitation. The cost after one trip is then $\left(1-q\right) r + 2qr = r\left(1+q\right)$, for a ratio of $1+q$, and after two trips the costs is $q\,(2r) + \left(1-q\right)(3r) = \left(3-q\right) r$, producing a ratio of $\left(3-q\right)/2$. Setting these to be equal yields $q = 1/3$, for a ratio of $4/3$.

If insufficient time is allowed for purchases before skiing, the best deterministic ratio is 2. Purchasing after the first ski trip with probability q (and after the second with probability $1-q$) leads to expected costs of $\left(1-q\right) r + 3qr = r\left(1+2q\right)$ after the first trip, and $\left(1-q\right)(2+2)\,r + 3qr = r\left(4-q\right)$, leading to a ratio of $2 - q/2$. Setting $1 + 2q = 2 - q/2$ yields $q = 2/5$, for a ratio of $9/5$.

More careful analysis, for which readers are referred to the references and the remainder of this volume, shows that the best achievable ratio approaches $\epsilon/(\epsilon-1) \approx 1.58197$ as p/r increases, approaching the limit from below if sufficient warning time is offered, and from above otherwise.

Applications

The primary initial results were directed towards problems of computer architecture; in particular, design questions for capacity conflicts in caches, and shared memory design in the presence of a shared communication channel. The motivation for these analyses was to find designs which would perform reasonably well on as-yet-unknown workloads, including those to be designed by competitors who may have chosen alternative designs which favor certain cases. While it is probably unrealistic to assume that precisely the least-desirable workloads will occur in ordinary practice, it is not unreasonable to assume that extremal workloads favoring either end of a decision will occur.

History and Further Reading

This technique of finding algorithms with bounded worst-case performance ratios is common in analyzing approximation algorithms. The initial proof techniques used for such analyses (the method of amortized analysis) were first presented by Sleator and Tarjan.

The reader is advised to consult the remainder of this volume for further extensions and applications of the principles of competitive on-line algorithms.

Cross References

- ▶ Algorithm DC-Tree for k Servers on Trees
- ▶ Metrical Task Systems
- ▶ Online List Update
- ▶ Online Paging and Caching
- ▶ Paging
- ▶ Work-Function Algorithm for k Servers

Recommended Reading

1. Karlin, A.R., Manasse, M.S., Rudolph, L., Sleator, D.D.: Competitive Snoopy Caching. Algorithmica **3**, 77–119 (1988) (Conference version: FOCS 1986, pp. 244–254)
2. Karlin, A.R., Manasse, M.S., McGeoch, L.A., Owicki, S.S.: Competitive Randomized Algorithms for Nonuniform Problems. Algorithmica **11**(6), 542–571 (1994) (Conference version: SODA 1990, pp. 301–309)
3. Reingold, N., Westbrook, J., Sleator, D.D.: Randomized Competitive Algorithms for the List Update Problem. Algorithmica **11**(1), 15–32 (1994) (Conference version included author Irani, S.: SODA 1991, pp. 251–260)

Slicing Floorplan Orientation

1983; Stockmeyer

EVANGELINE F. Y. YOUNG
Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, China

Keywords and Synonyms

Shape curve computation

Problem Definition

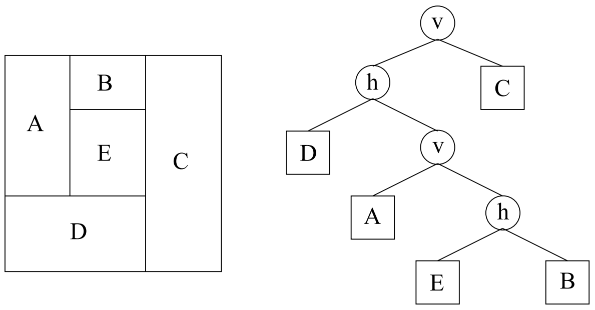
This problem is about finding the optimal orientations of the cells in a slicing floorplan to minimize the total area. In a floorplan, cells represent basic pieces of the circuit which are regarded as indivisible. After performing an initial placement, for example, by repeated application of a min-cut partitioning algorithm, the relative positions between the cells on a chip are fixed. Various optimization can then be done on this initial layout to optimize different cost measures such as chip area, interconnect length, routability, etc. One such optimization, as mentioned in Lauther [3], Otten [4], and Zibert and Saal [13], is to determine the best orientation of each cell to minimize the total chip area. This work by Stockmeyer [8] gives a polynomial time algorithm to solve the problem optimally in a special type of floorplans called *slicing floorplans* and shows that this orientation optimization problem in general non-slicing floorplans is NP-complete.

Slicing Floorplan

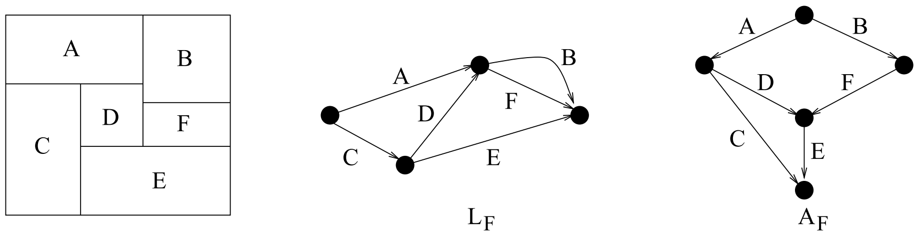
A floorplan consists of an enclosing rectangle subdivided by horizontal and vertical line segments into a set of non-overlapping *basic rectangles*. Two different line segments can meet but not cross. A floorplan F is characterized by a pair of planar acyclic directed graphs A_F and L_F defined as follows. Each graph has one source and one sink. The graph A_F captures the "above" relationships and has a vertex for each horizontal line segment, including the top and the bottom of the enclosing rectangle. For each basic rectangle R, there is an edge e_R directed from segment σ to segment σ' if and only if σ (or part of σ) is the top of R and σ' (or part of σ') is the bottom of R. There is a one-to-one correspondence between the basic rectangles and the edges in A_F. The graph L_F is defined similarly for the "left" relationships of the vertical segments. An example is shown in Fig. 1. Two floorplans F and G are equivalent if and only if $A_F = A_G$ and $L_F = L_G$. A floorplan F is slicing if and only if both its A_F and L_F are series parallel.

Slicing Tree

A slicing floorplan can also be described naturally by a rooted binary tree called *slicing tree*. In a slicing tree, each internal node is labeled by either an h or a v, indicating a horizontal or a vertical slice respectively. Each leaf corresponds to a basic rectangle. An example is shown in Fig. 2. There can be several slicing trees describing the same slicing floorplan but this redundancy can be removed by requiring the label of an internal node to differ from that of its right child [12]. For the algorithm presented in this work, a tree of smallest depth should be chosen and this depth minimization process can be done in $O(n \log n)$ time using the algorithm by Golumbic [2].



Slicing Floorplan Orientation, Figure 2
A slicing floorplan *F* and its slicing tree representation



Slicing Floorplan Orientation, Figure 1
A floorplan *F* and its A_F and L_F representing the above and left relationships

Orientation Optimization

In optimization of a floorplan layout, some freedom in moving the line segments and in choosing the dimensions of the rectangles are allowed. In the input, each basic rectangle R has two positive integers a_R and b_R, representing the dimensions of the cell that will be fit into R. Each cell has two possible orientations resulting in either the side of length a_R or b_R being horizontal. Given a floorplan F and an orientation ρ, each edge e in A_F and L_F is given a label $l(e)$ representing the height or the width of the cell corresponding to e depending on its orientation. Define an (F, ρ)-*placement* to be a labeling l of the vertices in A_F and L_F such that (i) the sources are labeled by zero, and (ii) if e is an edge from vertex σ to σ', $l(\sigma') \geq l(\sigma) + l(e)$. Intuitively, if σ is a horizontal segment, $l(\sigma)$ is the distance of σ from the top of the enclosing rectangle and the inequality constraint ensures that the basic rectangle corresponding to e is tall enough for the cell contained in it, and similarly for the vertical segments. Now, $h_F(\rho)$ (resp. $w_F(\rho)$) is defined to be the minimum label of the sink in $A_F(\rho)$ (resp. $L_F(\rho)$) over all (F, ρ)-placements, where $A_F(\rho)$ (resp. $L_F(\rho)$) is obtained from A_F (resp. L_F) by labeling the edges and vertices as described above. Intuitively, $h_F(\rho)$ and $w_F(\rho)$ give the minimum height and width of a floorplan F given an orientation ρ of all the cells such that each cell fits well into its associated basic rectangle. The orientation optimization problem can be defined formally as follows:

Problem 1 (Orientation Optimization Problem for Slicing Floorplan) INPUT: *A slicing floorplan F of n cells described by a slicing tree T, the widths and heights of the cells a_i and b_i for $i = 1 \ldots n$ and a cost function $\psi(h, w)$.*
OUTPUT: *An orientation ρ of all the cells that minimizes the objective function $\psi(h_F(\rho), w_F(\rho))$ over all orientations ρ.*

For this problem, Lauther [3] has suggested a greedy heuristic. Zibert and Saal [13] use integer programming methods to do rotation optimization and several other optimization simultaneously for general floorplans. In the following sections, an efficient algorithm will be given to solve the problem optimally in $O(nd)$ time where n is the number of cells and d is the depth of the given slicing tree.

Key Results

In the following algorithm, $F(u)$ denotes the floorplan described by the subtree rooted at u in the given slicing tree T and let $L(u)$ be the set of leaves in that subtree. For each node u of T, the algorithm constructs recursively a list of pairs:

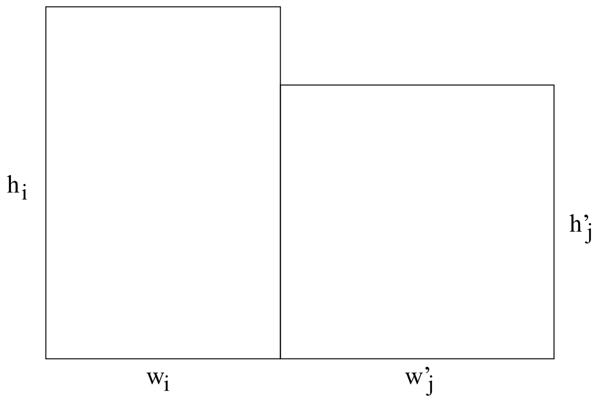
$$\{(h_1, w_1), (h_2, w_2), \ldots, (h_m, w_m)\}$$

where (1) $m \leq |L(u)| + 1$, (2) $h_i > h_{i+1}$ and $w_i < w_{i+1}$ for $i = 1 \ldots m - 1$, (3) there is an orientation ρ of the cells in $L(u)$ such that $(h_i, w_i) = (h_{F(u)}(\rho), w_{F(u)}(\rho))$ for each $i = 1 \ldots m$, and (4) for each orientation ρ of the cells in $L(u)$, there is a pair (h_i, w_i) in the list such that $h_i \leq h_{F(u)}(\rho)$ and $w_i \leq w_{F(u)}(\rho)$.

$L(u)$ is thus a non-redundant list of all possible dimensions of the floorplan described by the subtree rooted at u. Since the cost function ψ is non-decreasing, it can be minimized over all orientations by finding the minimum $\psi(h_i, w_i)$ over all the pairs (h_i, w_i) in the list constructed at the root of T. At the beginning, a list is constructed at each leaf node of T representing the possible dimensions of the cell. If a leaf cell has dimensions a and b with $a > b$, the list is $\{(a, b), (b, a)\}$. If $a = b$, there will just be one pair (a, b) in the list. (If the cell has a fixed orientation, there will also be just one pair as defined by the fixed orientation.) Notice that the condition (1) above is satisfied in these leaf node lists. The algorithm then works its way up the tree and constructs the list at each node recursively. In general, assume that u is an internal node with children v and v' and u represents a vertical slice. Let $\{(h_1, w_1) \ldots (h_k, w_k)\}$ and $\{(h'_1, w'_1) \ldots (h'_m, w'_m)\}$ be the lists at v and v' respectively where $k \leq |L(v)| + 1$ and $m \leq |L(v')| + 1$. A pair (h_i, w_i) from v can be put together by a vertical slice with a pair (h'_j, w'_j) from v' to give a pair:

$$join((h_i, w_i), (h'_j, w'_j)) = (\max(h_i, h'_j), w_i + w'_j)$$

in the list of u (see Fig. 3). The key fact is that most of the km pairs are sub-optimal and do not need to be considered. For example, if $h_i > h'_j$, there is no need to join



Slicing Floorplan Orientation, Figure 3
An illustration of the merging step

(h_i, w_i) with (h'_z, w'_z) for any $z > j$ since

$$\max(h_i, h'_z) = \max(h_i, h'_j) = h_i \,,$$
$$w_i + w'_z > w_i + w'_j$$

Similarly, if node u represents a horizontal slice, the *join* operation will be:

$$join((h_i, w_i), (h'_j, w'_j)) = (h_i + h'_j, \max(w_i, w'_j))$$

The algorithm also keeps two pointers for each element in the lists in order to construct back the optimal orientation at the end. The algorithm is summarized by the following pseudocode:

Pseudocode *Stockmeyer()*

```
1.  Initialize the list at each leaf node.
2.  Traverse the tree in postorder. At each internal node
    $u$ with children $v$ and $v'$, construct a list at node $u$ as
    follows:
3.      Let $\{(h_1, w_1) \ldots (h_k, w_k)\}$ and $\{(h'_1, w'_1) \ldots (h'_m, w'_m)\}$ be the lists at $v$ and $v'$ respectively.
4.      Initialize $i$ and $j$ to one.
5.      If $i > k$ or $j > m$, the whole list at $u$ is constructed.
6.      Add $join((h_i, w_i), (h'_j, w'_j))$ to the list with pointers pointing to $(h_i, w_i)$ and $(h'_j, w'_j)$ in $L(v)$ and $L(v')$ respectively.
7.      If $h_i > h'_j$, increment $i$ by 1.
8.      If $h_i < h'_j$, increment $j$ by 1.
9.      If $h_i = h'_j$, increment both $i$ and $j$ by 1.
10.     Go to step 5.
11. Compute $\psi(h_i, w_i)$ for each pair $(h_i, w_i)$ in the list $L_r$
    at the root $r$ of $T$.
12. Return the minimum $\psi(h_i, w_i)$ for all $(h_i, w_i)$ in $L_r$ and
    construct back the optimal orientation by following
    the pointers.
```

Correctness

The algorithm is correct since at each node u, a list is constructed that records all the possible non-redundant dimensions of the floorplan described by the subtree rooted at u. This can be proved easily by induction starting from the leaf nodes and working up the tree recursively. Since the cost function ψ is non-decreasing, it can be minimized over all orientations of the cells by finding the minimum $\psi(h_i, w_i)$ over all the pairs (h_i, w_i) in the list L_r constructed at the root r of T.

Runtime

At each internal node u with children v and v'. If the lengths of the lists at v and v' are k and m respectively, the time spent at u to combine the two lists is $O(k + m)$. Each possible dimension of a cell will thus invoke one unit of execution time at each node on its path up to the root in the post-order traversal. The total runtime is thus $O(d \times N)$ where N is the total number of realizations of all the n cells, which is equal to $2n$ in the orientation optimization problem. Therefore, the runtime of this algorithm is $O(nd)$.

Theorem 1 *Let $\psi(h, w)$ be non-decreasing in both arguments, i. e., if $h \le h'$ and $w \le w'$, $\psi(h, w) \le \psi(h', w')$, and computable in constant time. For a slicing floorplan F described by a binary slicing tree T, the problem of minimizing $\psi(h_F(\rho), w_F(\rho))$ over all orientations ρ can be solved in time $O(nd)$ time, where n is the number of leaves of T (equivalently, the number of cells of F) and d is the depth of T.*

Applications

Floorplan design is an important step in the physical design of VLSI circuits. Stockmeyer's optimal orientation algorithm [8] has been generalized to solve the area minimization problem in slicing floorplans [7], in hierarchical non-slicing floorplans of order five [6,9] and in general floorplans [5]. The floorplan area minimization problem is similar except that each *soft cell* now has a number of possible realizations, instead of just two different orientations. The same technique can be applied immediately to solve optimally the area minimization problem for slicing floorplans in $O(nd)$ time where n is the total number of realizations of all the cells in a given floorplan F and d is the depth of the slicing tree of F. Shi [7] has further improved this result to $O(n \log n)$ time. This is done by storing the list of non-redundant pairs at each node in a balanced binary search tree structure called *realization tree* and using a new merging algorithm to combine two such trees to create a new one. It is also proved in [7] that this $O(n \log n)$ time complexity is the lower bound for this area minimization problem in slicing floorplans.

For hierarchical non-slicing floorplans, Pan et al. [6] prove that the problem is NP-complete. Branch-and-bound algorithms are developed by Wang and Wong [9], and pseudopolynomial time algorithms are developed by Wang and Wong [10], and Pan et al. [6]. For general floorplans, Stockmeyer [8] has shown that the problem is strongly NP-complete. It is therefore unlikely to have any pseudopolynomial time algorithm. Wimer et al. [11], and Chong and Sahni [1] propose branch-and-bound algorithms. Pan et al. [5] develop algorithms for general floorplans that are approximately slicing.

Recommended Reading

1. Chong, K., Sahni, S.: Optimal Realizations of Floorplans. In: IEEE Trans. Comput. Aided Des. **12**(6), 793–901 (1993)
2. Golumbic, M.C.: Combinatorial Merging. IEEE Trans. Comput. **C-25**, 1164–1167 (1976)
3. Lauther, U.: A Min-Cut Placement Algorithm for General Cell Assemblies Based on a Graph Representation. J. Digital Syst. **4**, 21–34 (1980)
4. Otten, R.H.J.M.: Automatic Floorplan Design. In: Proceedings of the 19th Design Automation Conference, pp. 261–267 (1982)
5. Pan, P., Liu, C.L.: Area Minimization for Floorplans. In: IEEE Trans. Comput. Aided Des. **14**(1), 123–132 (1995)
6. Pan, P., Shi, W., Liu, C.L.: Area Minimization for Hierarchical Floorplans. In: Algorithmica **15**(6), 550–571 (1996)
7. Shi, W.: A Fast Algorithm for Area Minimization of Slicing Floorplan. In: IEEE Trans. Comput. Aided Des. **15**(12), 1525–1532 (1996)
8. Stockmeyer, L.: Optimal Orientations of Cells in Slicing Floorplan Designs. Inf. Control **59**, 91–101 (1983)
9. Wang, T.C., Wong, D.F.: Optimal Floorplan Area Optimization. In: IEEE Trans. Comput. Aided Des. **11**(8), 992–1002 (1992)
10. Wang, T.C., Wong, D.F.: A Note on the Complexity of Stockmeyer's Floorplan Optimization Technique. In: Algorithmic Aspects of VLSI Layout, Lecture Notes Series on Computing, vol. 2, pp. 309–320 (1993)
11. Wimer, S., Koren, I., Cederbaum, I.: Optimal Aspect Ratios of Building Blocks in VLSI. IEEE Trans. Comput. Aided Des. **8**(2), 139–145 (1989)
12. Wong, D.F., Liu, C.L.: A New Algorithm for Floorplan Design. Proceedings of the 23rd ACM/IEEE Design Automation Conference, pp. 101–107 (1986)
13. Zibert, K., Saal, R.: On Computer Aided Hybrid Circuit Layout. Proceedings of the IEEE Intl. Symp. on Circuits and Systems, pp. 314–318 (1974)

Snapshots in Shared Memory

1993; Afek, Attiya, Dolev, Gafni, Merritt, Shavit

ERIC RUPPERT
Department of Computer Science and Engineering, York University, Toronto, ON, Canada

Keywords and Synonyms

Atomic scan

Problem Definition

Implementing a snapshot object is an abstraction of the problem of obtaining a consistent view of several shared variables while other processes are concurrently updating those variables.

In an asynchronous shared-memory distributed system, a collection of n processes communicate by accessing shared data structures, called *objects*. The system provides basic types of shared objects; other needed types must be built from them. One approach uses locks to guarantee exclusive access to the basic objects, but this approach is not fault-tolerant, risks deadlock or livelock, and causes delays when a process holding a lock runs slowly. Lock-free algorithms avoid these problems but introduce new challenges. For example, if a process reads two shared objects, the values it reads may not be consistent if the objects were updated between the two reads.

A *snapshot object* stores a vector of m values, each from some domain D. It provides two operations: scan and update(i, v), where $1 \leq i \leq m$ and $v \in D$. If the operations are invoked sequentially, an update(i, v) operation changes the value of the ith component of the stored vector to v, and a scan operation returns the stored vector.

Correctness when snapshot operations by different processes overlap in time is described by the *linearizability* condition, which says operations should appear to occur instantaneously. More formally, for every execution, one can choose an instant of time for each operation (called its *linearization point*) between the invocation and the completion of the operation. (An incomplete operation may either be assigned no linearization point or given a linearization point at any time after its invocation.) The responses returned by all completed operations in the execution must return the same result as they would if all operations were executed sequentially in the order of their linearization points.

An implementation must also satisfy a progress property. *Wait-freedom* requires that each process completes each scan or update in a finite number of its own steps. The weaker *non-blocking* progress condition says the system cannot run forever without some operation completing.

This article describes implementations of snapshots from more basic types, which are also linearizable, without locks. Two types of snapshots have been studied. In a *single-writer* snapshot, each component is owned by a process, and only that process may update it. (Thus, for single-writer snapshots, $m = n$.) In a *multi-writer* snapshot, any process may update any component. There also exist algorithms for *single-scanner* snapshots, where only one process may scan at a time [10,13,14,16]. Snapshots were introduced by Afek et al. [1], Anderson [2] and Aspnes and Herlihy [4].

Space complexity is measured by the number of basic objects used and their size (in bits). Time complexity is measured by the maximum number of steps a process must do to finish a scan or update, where a step is an access to a basic shared object. (Local computation and local memory accesses are usually not counted.) Complexity bounds will be stated in terms of n, m, $d = \log|D|$ and k, the number of operations invoked in an execution. Ordinarily, there is no bound on k.

Most of the algorithms below use read-write registers, the most elementary shared object type. A *single-writer* register may only be written by one process. A *multi-writer* register may be written by any process. Some algorithms using stronger types of basic objects are discussed in Sect. "Wait-Free Implementations from Small, Stronger Objects".

Key Results

A Simple Non-blocking Implementation from Small Registers

Suppose each component of a single-writer snapshot object is represented by a single-writer register. Process i does an update(i, v) by writing v and a sequence number into register i, and incrementing its sequence number. Performing a scan operation is more difficult than merely reading each of the m registers, since some registers might change while these reads are done. To scan, a process repeatedly reads all the registers. A sequence of reads of all the registers is called a *collect*. If two collects return the same vector, the scan returns that vector (with the sequence numbers stripped away). The sequence numbers ensure that, if the same value is read in a register twice, the register had that value during the entire interval between the two reads. The scan can be assigned a linearization point between the two identical collects, and updates are linearized at the write. This algorithm is non-blocking, since a scan continues running only if at least one update operation is completed during each collect. A similar algorithm, with process identifiers appended to the sequence numbers, implements a non-blocking multi-writer snapshot from m multi-writer registers.

Wait-Free Implementations from Large Registers

Afek et al. [1] described how to modify the non-blocking single-writer snapshot algorithm to make it wait-free using scans embedded within the updates. An update(i, v) first does a scan and then writes a triple containing the scan's result, v and a sequence number into register i. While a process P is repeatedly performing collects to do a scan, either two collects return the same vector (which P can return) or P will eventually have seen three different triples in the register of some other process. In the latter case, the third triple that P saw must contain a vector that is the result of a scan that started after P's scan, so P's scan outputs that vector. Updates and scans that terminate after seeing two identical collects are assigned linearization points as before. If one scan obtains its output from an embedded scan, the two scans are given the same linearization point. This is a wait-free single-writer snapshot implementation from n single-writer registers of $(n+1)d + \log k$ bits each. Operations complete within $O(n^2)$ steps. Afek et al. [1] also describe how to replace the unbounded sequence numbers with handshaking bits. This requires $n\Theta(nd)$-bit registers and n^2 1-bit registers. Operations still complete in $O(n^2)$ steps.

The same idea can be used to build multi-writer snapshots from multi-writer registers. Using unbounded sequence numbers yields a wait-free algorithm that uses m registers storing $\Theta(nd + \log k)$ bits each, in which each operation completes within $O(mn)$ steps. (This algorithm is given explicitly in [9].) No algorithm can use fewer than m registers if $n \geq m$ [9]. If handshaking bits are used instead, the multi-writer snapshot algorithm uses n^2 1-bit registers, $m(d + \log n)$-bit registers and n (md)-bit registers, and each operation uses $O(nm + n^2)$ steps [1].

Guerraoui and Ruppert [12] gave a similar wait-free multi-writer snapshot implementation that is anonymous, *i. e.*, it does not use process identifiers and all processes are programmed identically.

Anderson [3] gave an implementation of a multi-writer snapshot from a single-writer snapshot. Each process stores its latest update to each component of the multi-writer snapshot in the single-writer snapshot, with associated timestamp information computed by scanning the single-writer snapshot. A scan is done using just one scan of the single-writer snapshot. An update requires scanning and updating the single-writer snapshot twice. The implementation involves some blow-up in the size of the components, *i. e.*, to implement a multi-writer snapshot with domain D requires a single-writer snapshot with a much larger domain D'. If the goal is to implement multi-writer snapshots from single-writer registers (rather than multi-writer registers), Anderson's construction gives a more efficient solution than that of Afek et al.

Attiya, Herlihy and Rachman [7] defined the *lattice agreement* object, which is very closely linked to the problem of implementing a single-writer snapshot when there is a known upper bound on k. Then, they showed how to construct a single-writer snapshot (with no bound on k) from an infinite sequence of lattice agreement objects. Each snapshot operation accesses the lattice agreement object twice and does $O(n)$ additional steps. Their implementations of lattice agreement are discussed in Sect. "Wait-Free Implementations from Small, Stronger Objects".

Attiya and Rachman [8] used a similar approach to give a single-writer snapshot implementation from large single-writer registers using $O(n \log n)$ steps per operation. Each update has an associated sequence number. A scanner traverses a binary tree of height $\log k$ from root to leaf (here, a bound on k is required). Each node has an array of n single-writer registers. A process arriving at a node writes its current vector into a single-writer register associated with the node and then gets a new vector by combining information read from all n registers. It proceeds to the left or right child depending on the sum of the sequence numbers in this vector. Thus, all scanners can be linearized in the order of the leaves they reach. Updates are performed by doing a similar traversal of the tree. The bound on k can be removed as in [7]. Attiya and Rachman also give a more direct implementation that achieves this by recycling the snapshot object that assumes a bound on k. Their algorithm has also been adapted to solve condition-based consensus [15].

Attiya, Fouren and Gafni [6] described how to adapt the algorithm of Attiya and Rachman [8] so that the number of steps required to perform an operation depends on the number of processes that actually access the object, rather than the number of processes in the system.

Attiya and Fouren [5] solve lattice agreement in $O(n)$ steps. (Here, instead of using the terminology of lattice agreement, the algorithm is described in terms of implementing a snapshot in which each process does at most one snapshot operation.) The algorithm uses, as a data structure, a two-dimensional array of $O(n^2)$ *reflectors*. A reflector is an object that can be used by two processes to exchange information. Each reflector is built from two large single-writer registers. Each process chooses a path through the array of reflectors, so that at most two processes visit each reflector. Each reflector in column i is used by process i to exchange information with one process $j < i$. If process i reaches the reflector first, process j learns about i's update (if any). If process j reaches it first, then process i learns all the information that j has already gathered. (If both reach it at about the same time, both processes learn the information described above.) As the processes move from column $i - 1$ to column i, a process that enters column i at some row r will have gathered all the information that has been gathered by any process that enters column i below row r (and possibly more). This invariant is maintained by ensuring that if process i passes information to any process $j < i$ in row r of column i, it also passes that information to all processes that entered column i above row r. Furthermore, process i exits its column i at a row that matches the amount of information it learns while traveling through the column. When processes have reached the rightmost column of the array, the ones in higher rows know strictly more than the ones in lower rows. Thus, the linearization order of their scans is the order in which they exit the rightmost column, from bottom to top. The techniques of Attiya, Herlihy and Rachman [7,8], mentioned above, can be used to remove the restriction that each process performs at most one operation. The number of steps per operation is still $O(n)$.

Wait-Free Implementations from Small, Stronger Objects

All of the wait-free implementations described above use registers that can store $\Omega(m)$ bits each, and are therefore not practical when m is large. Some implementations from smaller objects equipped with stronger synchronization operations, rather than just reads and writes, are described in this section. An object is considered to be small if it can store $O(d + \log n + \log k)$ bits. This means that it can store a constant number of component values, process identifiers and sequence numbers.

Attiya, Herlihy and Rachman [7] gave an elegant divide-and-conquer recursive solution to the lattice agreement problem. The division of processes into groups for the recursion can be done dynamically using test&set objects. This provides a snapshot algorithm that runs in $O(n)$ time per operation, and uses $O(kn^2 \log n)$ small single-writer registers and $O(kn \log^2 n)$ test&set objects. (This requires modifying their implementation to replace those registers that are large, which are written only once, by many small registers.) Using randomization, each test&set object can be replaced by single-writer registers to give a snapshot implementation from registers only with $O(n)$ expected steps per operation.

Jayanti [13] gave a multi-writer snapshot implementation from $O(mn^2)$ small compare&swap objects where updates take $O(1)$ steps and scans take $O(m)$ steps. He began with a very simple single-scanner, single-writer snapshot implementation from registers that uses a secondary array to store a copy of recent updates. A scan clears that array, collects the main array, and then collects the secondary array to find any overlooked updates. Several additional mechanisms are introduced for the general, multi-writer, multi-scanner snapshot. In particular, compare&swap operations are used instead of writes to coordinate writers updating the same component and multiple scanners coordinate with one another to simulate a single scanner. Jayanti's algorithm builds on an earlier paper by Riany, Shavit and Touitou [16], which gave an implementation that achieved similar complexity, but only for a single-writer snapshot.

Applications

Applications of snapshots include distributed databases, storing checkpoints or backups for error recovery, garbage collection, deadlock detection, debugging distributed programmes and obtaining a consistent view of the values reported by several sensors. Snapshots have been used as building blocks for distributed solutions to randomized consensus and approximate agreement. They are also helpful as a primitive for building other data structures. For example, consider implementing a counter that stores an integer and provides increment, decrement and read operations. Each process can store the number of increments it has performed minus the number of its decrements in its own component of a single-writer snapshot object, and the counter may be read by summing the values from a scan. See [10] for references on many of the applications mentioned here.

Open Problems

Some complexity lower bounds are known for implementations from registers [9], but there remain gaps between the best known algorithms and the best lower bounds. In particular, it is not known whether there is an efficient wait-free implementation of snapshots from small registers.

Experimental Results

Riany, Shavit and Touitou gave performance evaluation results for several implementations [16].

Cross References

► Implementing Shared Registers in Asynchronous Message-Passing Systems
► Linearizability
► Registers

Recommended Reading

See also Fich's survey paper on the complexity of implementing snapshots [11].

1. Afek, Y., Attiya, H., Dolev, D., Gafni, E., Merritt, M., Shavit, N.: Atomic snapshots of shared memory. J. Assoc. Comput. Mach. **40**, 873–890 (1993)
2. Anderson, J.H.: Composite registers. Distrib. Comput. **6**, 141–154 (1993)
3. Anderson, J.H.: Multi-writer composite registers. Distrib. Comput. **7**, 175–195 (1994)
4. Aspnes, J., Herlihy, M.: Wait-free data structures in the asynchronous PRAM model. In: Proc. 2nd ACM Symposium on Parallel Algorithms and Architectures, Crete, July 1990. pp. 340–349. ACM, New York, 1990
5. Attiya, H., Fouren, A.: Adaptive and efficient algorithms for lattice agreement and renaming. SIAM J. Comput. **31**, 642–664 (2001)
6. Attiya, H., Fouren, A., Gafni, E.: An adaptive collect algorithm with applications. Distrib. Comput. **15**, 87–96 (2002)
7. Attiya, H., Herlihy, M., Rachman, O.: Atomic snapshots using lattice agreement. Distrib. Comput. **8**, 121–132 (1995)
8. Attiya, H., Rachman, O.: Atomic snapshots in $O(n \log n)$ operations. SIAM J. Comput. **27**, 319–340 (1998)
9. Ellen, F., Fatourou, P., Ruppert, E.: Time lower bounds for implementations of multi-writer snapshots. J. Assoc. Comput. Mach. **54**(6) article 30 (2007)
10. Fatourou, P., Kallimanis, N.D.: Single-scanner multi-writer snapshot implementations are fast! In: Proc. 25th ACM Symposium on Principles of Distrib. Comput. Colorado, July 2006 pp. 228–237. ACM, New York (2006)
11. Fich, F.E.: How hard is it to take a snapshot? In: SOFSEM 2005: Theory and Practice of Computer Science. Liptovský Ján, January 2005, LNCS, vol. 3381, pp. 28–37. Springer (2005)
12. Guerraoui, R., Ruppert, E.: Anonymous and fault-tolerant shared-memory computing. Distrib. Comput. **20**(3) 165–177 (2007)
13. Jayanti, P.: An optimal multi-writer snapshot algorithm. In: Proc. 37th ACM Symposium on Theory of Computing. Baltimore, May 2005, pp. 723–732. ACM, New York (2005)
14. Kirousis, L.M., Spirakis, P., Tsigas, P.: Simple atomic snapshots: A linear complexity solution with unbounded time-stamps. Inf. Process. Lett. **58**, 47–53 (1996)
15. Mostéfaoui, A., Rajsbaum, S., Raynal, M., Roy, M.: Condition-based consensus solvability: a hierarchy of conditions and efficient protocols. Distrib. Comput. **17**, 1–20 (2004)
16. Riany, Y., Shavit, N., Touitou, D.: Towards a practical snapshot algorithm. Theor. Comput. Sci. **269**, 163–201 (2001)

Sojourn Time

► Minimum Flow Time
► Shortest Elapsed Time First Scheduling

Sorting of Multi-Dimensional Keys

► String Sorting

Sorting Signed Permutations by Reversal (Reversal Distance)

2001; Bader, Moret, Yan

DAVID A. BADER
College of Computing, Georgia Institute of Technology, Atlanta, GA, USA

Keywords and Synonyms

Sorting by reversals; Inversion distance; Reversal distance

Problem Definition

This entry describes algorithms for finding the minimum number of steps needed to sort a signed permutation (also known as: inversion distance, reversal distance). This is a real-world problem and for example is used in computational biology.

Inversion distance is a difficult computational problem that has been studied intensively in recent years [1,4,6,7,8,9,10]. Finding the inversion distance between unsigned permutations is NP-hard [7], but with signed ones, it can be done in linear time [1].

Key Results

Bader et al. [1] present the first worst-case linear-time algorithm for computing the reversal distance that is simple and practical and runs faster than previous methods. Their key innovation is a new technique to compute connected components of the overlap graph using only a stack, which results in the simple linear-time algorithm for computing the inversion distance between two signed permutations. Bader et al. provide ample experimental evidence that their linear-time algorithm is efficient in practice as well as in theory: they coded it as well as the algorithm of Berman and Hannenhalli, using the best principles of algorithm engineering to ensure that both implementations would be as efficient as possible, and compared their running times on a large range of instances generated through simulated evolution.

Bafna and Pevzner introduced the cycle graph of a permutation [3], thereby providing the basic data structure for inversion distance computations. Hannenhalli and Pevzner then developed the basic theory for expressing the inversion distance in easily computable terms (number of breakpoints minus number of cycles plus number of hurdles plus a correction factor for a fortress [3,15]—hurdles and fortresses are easily detectable from a connected component analysis). They also gave the first polynomial-time algorithm for sorting signed permutations by reversals [9]; they also proposed a $O(n^4)$ implementation of their algorithm which runs in quadratic time when restricted to distance computation. Their algorithm requires the computation of the connected components of the overlap graph, which is the bottleneck for the distance computation. Berman and Hannenhalli later exploited some combinatorial properties of the cycle graph to give a $O(n\alpha(n))$ algorithm to compute the connected components, leading to a $O(n^2\alpha(n))$ implementation of the sorting algorithm [6], where α is the inverse Ackerman function. (The later Kaplan–Shamir–Tarjan (KST) algorithm [10] reduces the time needed to compute the shortest sequence of inversions, but uses the same algorithm for computing the length of that sequence.)

No algorithm that actually builds the overlap graph can run in linear time, since that graph can be of quadratic size. Thus, Bader's key innovation is to construct an *overlap forest* such that two vertices belong to the same tree in the forest exactly when they belong to the same connected component in the overlap graph. An overlap forest (the composition of its trees is unique, but their structure is arbitrary) has exactly one tree per connected component of the overlap graph and is thus of linear size. The linear-time step for computing the connected components scans the permutation twice. The first scan sets up a trivial forest in which each node is its own tree, labeled with the beginning of its cycle. The second scan carries out an iterative refinement of this first forest, by adding edges and so merging trees in the forest; unlike a Union-Find, however, this algorithm does not attempt to maintain the trees within certain shape parameters. This step is the key to Bader's linear-time algorithm for computing the reversal distance between signed permutations.

Applications

Some organisms have a single chromosome or contain single-chromosome organelles (such as mitochondria or chloroplasts), the evolution of which is largely independent of the evolution of the nuclear genome. Given a particular strand from a single chromosome, whether linear or circular, we can infer the ordering and directionality of the genes, thus representing each chromosome by an ordering of oriented genes. In many cases, the evolutionary process that operates on such single-chromosome organisms consists mostly of inversions of portions of the chromosome; this finding has led many biologists to reconstruct phylogenies based on gene orders, using as a measure of evolutionary distance between two genomes the inversion distance, i. e., the smallest number of inversions needed to transform one signed permutation into the other [11,12,14].

The linear-time algorithm is in wide-use (as it has been cited nearly 200 times within the first several years of its publication). Examples include the handling multichromosomal genome rearrangements [16], genome comparison [5], parsing RNA secondary structure [13], and phylogenetic study of the HIV-1 virus [2].

Open Problems

Efficient algorithms for computing minimum distances with weighted inversions, transpositions, and inverted transpositions, are open.

Experimental Results

Bader et al. give experimental results in [1].

URL to Code

An implementation of the linear-time algorithm is available as C code from www.cc.gatech.edu/~bader. Two other dominated implementations are available that are designed to compute the shortest sequence of inversions as well as its length; one, due to Hannenhalli that implements his first algorithm [9], which runs in quadratic time when computing distances, while the other, a Java applet written by Mantin (http://www.math.tau.ac.il/~rshamir/GR/) implements the KST algorithm [10], but uses an explicit representation of the overlap graph and thus also takes quadratic time. The implementation due to Hannenhalli is very slow and implements the original method of Hannenhalli and Pevzner and not the faster one of Berman and Hannenhalli. The KST applet is very slow as well since it explicitly constructs the overlap graph.

Cross References

For finding the actual sorting sequence, see the entry:
► Sorting Signed Permutations by Reversal (Reversal Sequence)

Recommended Reading

1. Bader, D.A., Moret, B.M.E., Yan, M.: A linear-time algorithm for computing inversion distance between signed permutations with an experimental study. J. Comput. Biol. **8**(5), 483–491 (2001) An earlier version of this work appeared In: the Proc. 7th Int'l Workshop on Algorithms and Data Structures (WADS 2001)
2. Badimo, A., Bergheim, A., Hazelhurst, S., Papathanasopolous, M., Morris, L.: The stability of phylogenetic tree construction of the HIV-1 virus using genome-ordering data versus env gene data. In: Proc. ACM Ann. Research Conf. of the South African institute of computer scientists and information technologists on enablement through technology (SAICSIT 2003), vol. 47, pp. 231–240, Fourways, ACM, South Africa, September 2003
3. Bafna, V., Pevzner, P.A.: Genome rearrangements and sorting by reversals. In: Proc. 34th Ann. IEEE Symp. Foundations of Computer Science (FOCS93), pp. 148–157. IEEE Press (1993)
4. Bafna, V., Pevzner, P.A.: Genome rearrangements and sorting by reversals. SIAM J. Comput. **25**, 272–289 (1996)
5. Bergeron, A., Stoye, J.: On the similarity of sets of permutations and its applications to genome comparison. J. Comput. Biol. **13**(7), 1340–1354 (2006)
6. Berman, P., Hannenhalli, S.: Fast sorting by reversal. In: Hirschberg, D.S., Myers, E.W. (eds.) Proc. 7th Ann. Symp. Combinatorial Pattern Matching (CPM96). Lecture Notes in Computer Science, vol. 1075, pp. 168–185. Laguna Beach, CA, June 1996. Springer (1996)
7. Caprara, A.: Sorting by reversals is difficult. In: Proc. 1st Conf. Computational Molecular Biology (RECOMB97), pp. 75–83. ACM, Santa Fe, NM (1997)
8. Caprara, A.: Sorting permutations by reversals and Eulerian cycle decompositions. SIAM J. Discret. Math. **12**(1), 91–110 (1999)
9. Hannenhalli, S., Pevzner, P.A.: Transforming cabbage into turnip (polynomial algorithm for sorting signed permutations by reversals). In: Proc. 27th Ann. Symp. Theory of Computing (STOC95), pp. 178–189. ACM, Las Vegas, NV (1995)
10. Kaplan, H., Shamir, R., Tarjan, R.E.: A faster and simpler algorithm for sorting signed permutations by reversals. SIAM J. Comput. **29**(3), 880–892 (1999) First appeared In: Proc.8th Ann. Symp. Discrete Algorithms (SODA97), pp. 344–351. ACM Press, New Orleans, LA
11. Olmstead, R.G., Palmer, J.D.: Chloroplast DNA systematics: a review of methods and data analysis. Am. J. Bot. **81**, 1205–1224 (1994)
12. Palmer, J.D.: Chloroplast and mitochondrial genome evolution in land plants. In: Herrmann, R. (ed.) Cell Organelles, pp. 99–133. Springer, Vienna (1992)
13. Rastegari, B., Condon, A.: Linear time algorithm for parsing RNA secondary structure. In: Casadio, R., Myers, E.: (eds.) Proc. 5th Workshop Algs. in Bioinformatics (WABI'05). Lecture Notes in Computer Science, vol. 3692, pp. 341–352. Springer, Mallorca, Spain (2005)
14. Raubeson, L.A., Jansen, R.K.: Chloroplast DNA evidence on the ancient evolutionary split in vascular land plants. Science **255**, 1697–1699 (1992)
15. Setubal, J.C., Meidanis, J.: Introduction to Computational Molecular Biology. PWS, Boston, MA (1997)
16. Tesler, G.: Efficient algorithms for multichromosomal genome rearrangements. J. Comput. Syst. Sci. **63**(5), 587–609 (2002)

Sorting Signed Permutations by Reversal (Reversal Sequence)

2004; Tannier, Sagot

ERIC TANNIER
NRIA Rhone-Alpes, University of Lyon, Lyon, France

Keywords and Synonyms

Sorting by inversions

Problem Definition

A *signed permutation* π of size n is a permutation over $\{-n, \dots, -1, 1 \dots n\}$, where $\pi_{-i} = -\pi_i$ for all i.

The *reversal* $\rho = \rho_{i,j}$ $(1 \le i \le j \le n)$ is an operation that reverses the order and flips the signs of the elements

$\pi_i, \ldots, \pi_j$ in a permutation π:

$$\pi \cdot \rho = (\pi_1, \ldots, \pi_{i-1}, -\pi_j, \ldots, -\pi_i, \pi_{j+1}, \ldots, \pi_n) .$$

If $\rho_1, \ldots, \rho_k$ is a sequence of reversals, it is said to *sort* a permutation π if $\pi \cdot \rho_1 \cdots \rho_k = Id$, where $Id = (1, \ldots, n)$ is the identity permutation. The length of a shortest sequence of reversals sorting π is called the *reversal distance* of π, and is denoted by $d(\pi)$.

If the computation of $d(\pi)$ is solved in linear time [2] (see the entry "reversal distance"), the computation of a sequence of size $d(\pi)$ that sorts π is more complicated and no linear algorithm is known so far. The best complexity is currently achieved by the solution of Tannier and Sagot [17], which has later been improved papers by Tannier, Bergeron and Sagot [18] and Han [8].

Key Results

Recall there is a linear algorithm to compute the reversal distance thanks to the formula $d(\pi) = n + 1 - c(\pi) + t(\pi)$ (notation from [4]), where $c(\pi)$ is the number of cycles in the breakpoint graph, and $t(\pi)$ is computed from the unoriented components of the permutation (see the entry "reversal distance"). Once this is known, there is a trivial algorithm that computes a sequence of size $d(\pi)$: try every possible reversal ρ at one step, until you find one such that $d(\pi \cdot \rho) = d(\pi) - 1$. Such a reversal is called *safe*. This necessitates $O(n)$ computations for every possible reversal (they are at most $(n + 1)(n + 2)/2 = O(n^2)$), and iterating this to find a sequence yields an $O(n^4)$ algorithm.

The first polynomial algorithm by Hannenhalli and Pevzner [9] was not achieving a better complexity and the algorithmic study of finding shortest sequences of reversals began its history.

The Scenario of Reversals

All the published solutions for the computations of a sorting sequence are divided into two, following the division of the distance formula into two parameters: a first part computes a sequence of reversals so that the resulting permutation has no unoriented component, and a second part sorts all oriented components.

The first part was given its best solution by Kaplan, Shamir and Tarjan [10], whose algorithm runs in linear time when coupled with the linear distance computation [2], and it is based on Hannenhalli and Pevzner's [9] early results.

The second part is the bottleneck of the whole procedure. At this point, if there is no unoriented component, the distance is $d(\pi) = n + 1 - c(\pi)$, so a safe reversal is one that increases $c(\pi)$ and do not create unoriented components (that would increase $t(\pi)$).

A reversal that increases $c(\pi)$ is called *oriented*. Finding an oriented reversal is an easy part: any two consecutive numbers that have different signs in the permutation define one. The hard part is to make sure it does not increase the number of unoriented components.

The quadratic algorithms designed on one side by Berman and Hannenhalli [5] and on the other by Kaplan, Shamir and Tarjan [10] are based on the linear recognition of safe reversals. No better algorithm is known so far to recognize safe reversals, and it seemed that a lower bound had been reached, as witnessed by a survey of Ozery-Flato and Shamir [14] in which they wrote that "a central question in the study of genome rearrangements is whether one can obtain a subquadratic algorithm for sorting by reversals". This was obtained by Tannier and Sagot [17], who proved that the recognition of safe reversal at each step is not necessary, but only the recognition of oriented reversals.

The algorithm is based on the following theorem, taken from [18]. A sequence of oriented reversals $\rho_1, \ldots, \rho_k$ is said to be *maximal* if there is no oriented reversal in $\pi \cdot \rho_1 \cdots \rho_k$. In particular a sorting sequence is maximal, but the converse is not true.

Theorem 1 *If S is a maximal but not a sorting sequence of oriented reversals for a permutation, then there exists a nonempty sequence S' of oriented reversals such that S may be split into two parts $S = S_1, S_2$, and S_1, S', S_2 is a sequence of oriented reversal.*

This allows to construct sequences of oriented reversals instead of safe reversals, and increase their size by adding reversals inside the sequence instead of at the end, and obtain a sorting sequence.

This algorithm, with a classical data structure to represent permutations (as an array for example) has still an $O(n^2)$ complexity, because at each step it has to test the presence of an oriented reversal, and apply it to the permutation.

The slight modification of a data structure invented by Kaplan and Verbin [11] allows to pick and apply an oriented reversal in $O(\sqrt{n \log n})$, and using this, Tannier and Sagot's algorithm achieves $O(n^{3/2} \sqrt{\log n})$ time complexity.

Recently, Han [8] announced another data structure that allows to pick and apply an oriented reversal in $O(\sqrt{n})$ time, and a similar slight modification can probably decrease the complexity of the overall method to $O(n^{3/2})$.

The Space of all Optimal Solutions

Almost all the studies on sorting sequences of reversals were devoted to giving only one sequence, though it has been remarked that there are often plenty of them (it may be over several millions even for $n \leq 10$). A few studies have tried to fill this deficiency.

An algorithm to enumerate all safe reversals at one step has been designed and implemented by Siepel [16]. A structure of the space of optimal solutions has been discovered by Chauve et al. [3], and the algorithmics related to this structure are studied in [6].

Applications

The motivation as well as the main application of this problem is in computational biology. Signed permutations are an adequate object to model the relative position and orientation of homologous blocks of DNA in two species. A generalization of this problem to multichromosomal models has been solved by and applied in mammalian genomes [15] to argue for a model of evolution where reversals do not occur randomly.

Ajana et al. [1] used a random exploration in the space of solutions to test the hypothesis that in bacteria, reversals occur mainly around an origin or terminus of replication.

Generalizations to the comparison of more than two genomes has been the subject of an abundant literature, and applied to reconstruct evolutionary events and the organization of the genomes of common ancestors of living species, or to infer gene orthology from their positions, and they are based on heuristic principles guided by the theory of sorting signed permutations by reversals [12,13].

Open Problems

- Finding a better complexity than $O(n^{3/2})$. It could be achieved by a smarter data structure, or changing the principle of the algorithm, so that there is no need to apply at each step a sorting reversal to be able to compute the next ones.
- The efficient representation and enumeration of the whole set of solutions (see some advances in [3,6]).
- Finding, among the solutions, the ones that fit some biological constraints, as preserving some common groups of genes or favoring small inversions (see some advances in [7]).

Experimental Results

The algorithm of Tannier, Bergeron and Sagot [18] has been implemented in its quadratic version (without any special data structure, which are probably worth only for very big sizes of permutations) by Diekmann (biomserv.univ-lyon1.fr/~tannier/PSbR/), but no implementation of the data structures nor experiments on the complexity are reported.

URL to Code

- www.cse.ucsd.edu/groups/bioinformatics/GRIMM/
  In Pevzner's group, Tesler has put online an implementation of the multicromosomal generalization of the algorithm of Kaplan, Shamir, and Tarjan [10], that he has called GRIMM, for "Genome Rearrangements In Man and Mouse".
- www.cs.unm.edu/~moret/GRAPPA/
  GRAPPA stands for "Genome Rearrangements Analysis under Parsimony and other Phylogenetic Algorithms". It contains the distance computation, and the algorithm to find all safe reversals at one step. It has been developed in Moret's team.
- www.math.tau.ac.il/~rshamir/GR/
  An applet written by Mantin implementing the algorithm of Kaplan, Shamir and Tarjan [10].
- biomserv.univ-lyon1.fr/~tannier/PSbR/
  A program by Diekmann to find a scenario of reversals with additional constraints for signed permutations, implementing the algorithm of Tannier and Sagot [17].
- www.geocities.com/mdvbraga/baobabLuna.html
  A program by Braga for the manipulation of permutations, and in particular sorting signed permutations by reversals, and giving a condensed representation of all optimal sorting sequences, implementing an algorithm of [6].

Cross References

► Sorting Signed Permutations by Reversal (Reversal Distance)

Recommended Reading

1. Ajana, Y., Lefebvre, J.-F., Tillier, E., El-Mabrouk, N.: Exploring the Set of All Minimal Sequences of Reversals – An Application to Test the Replication-Directed Reversal Hypothesis, Proceedings of the Second Workshop on Algorithms in Bioinformatics. Lecture Notes in Computer Science, vol. 2452, pp. 300–315. Springer, Berlin (2002)
2. Bader, D.A., Moret, B.M.E., Yan, M.: A Linear-Time Algorithm for Computing Inversion Distance between Signed Permutations with an Experimental Study. J. Comput. Biol. **8**(5), 483–491 (2001)
3. Bergeron, A., Chauve, C., Hartman, T., St-Onge, K.: On the properties of sequences of reversals that sort a signed permutation. Proceedings of JOBIM'02, 99–108 (2002)

4. Bergeron, A., Mixtacki, J., Stoye, J.: The inversion distance problem. In: Gascuel, O. (ed.) Mathematics of evolution and phylogeny. Oxford University Press, USA (2005)
5. Berman, P., Hannenhalli, S.: Fast Sorting by Reversal, proceedings of CPM '96. Lecture notes in computer science **1075**, 168–185 (1996)
6. Braga, M.D.V., Sagot, M.F., Scornavacca, C., Tannier, E.: The Solution Space of Sorting by Reversals. In: Proceedings of ISBRA'07. Lect. Notes Comp. Sci. **4463**, 293–304 (2007)
7. Diekmann, Y., Sagot, M.F., Tannier, E.: Evolution under Reversals: Parsimony and Conversation of Common Intervals. IEEE/ACM Transactions in Computational Biology and Bioinformatics, **4**, 301–309, 1075 (2007)
8. Han, Y.: Improving the Efficiency of Sorting by Reversals, Proceedings of The 2006 International Conference on Bioinformatics and Computational Biology. Las Vegas, Nevada, USA (2006)
9. Hannenhalli, S., Pevzner, P.: Transforming cabbage into turnip (polynomial algorithm for sorting signed permutations by reversals). J. ACM **46**, 1–27 (1999)
10. Kaplan, H., Shamir, R., Tarjan, R.E.: Faster and simpler algorithm for sorting signed permutations by reversals. SIAM J. Comput. **29**, 880–892 (1999)
11. Kaplan, H., Verbin, E.: Efficient data structures and a new randomized approach for sorting signed permutations by reversals. In: Proceedings of CPM'03. Lecture Notes in Computer Science **2676**, 170–185
12. Moret, B.M.E., Tang, J., Warnow, T.: Reconstructing phylogenies from gene-content and gene-order data. In: Gascuel, O. (ed.) Mathematics of Evolution and Phylogeny. pp. 321–352, Oxford Univ. Press, USA (2005)
13. Murphy, W., et al.: Dynamics of Mammalian Chromosome Evolution Inferred from Multispecies Comparative Maps. Science **309**, 613–617 (2005)
14. Ozery-Flato, M., Shamir, R.: Two notes on genome rearrangement. J. Bioinf. Comput. Biol. **1**, 71–94 (2003)
15. Pevzner, P., Tesler, G.: Human and mouse genomic sequences reveal extensive breakpoint reuse in mammalian evolution. PNAS **100**, 7672–7677 (2003)
16. Siepel, A.C.: An algorithm to enumerate sorting reversals for signed permutations. J. Comput. Biol. **10**, 575–597 (2003)
17. Tannier, E., Sagot, M.-F.: Sorting by reversals in subquadratic time. In: Proceedings of CPM'04. Lecture Notes Comput. Sci. **3109**, 1–13
18. Tannier, E., Bergeron, A., Sagot, M.-F.: Advances on Sorting by Reversals. Discret. Appl. Math. **155**, 881–888 (2006)

Sorting by Transpositions and Reversals (Approximate Ratio 1.5)

2004; Hartman, Sharan

CHIN LUNG LU
Institute of Bioinformatics & Department of Biological Science and Technology, National Chiao Tung University, Hsinchu, Taiwan

Keywords and Synonyms

Genome rearrangements

Problem Definition

One of the most promising ways to determine evolutionary distance between two organisms is to compare the order of appearance of identical (e. g., orthologous) genes in their genomes. The resulting genome rearrangement problem calls for finding a shortest sequence of rearrangement operations that sorts one genome into the other. In this work [8], Hartman and Sharan provide a 1.5-approximation algorithm for the problem of sorting by transpositions, transreversals and revrevs, improving on a previous 1.75 ratio for this problem. Their algorithm is also faster than current approaches and requires $O(n^{3/2}\sqrt{\log n})$ time for n genes.

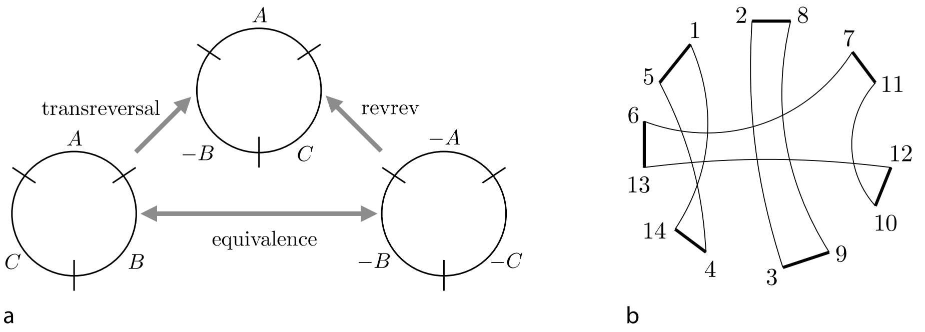
Notations and Definition

A *signed permutation* $\pi = [\pi_1, \pi_2, \dots, \pi_n]$ on $n(\pi) \equiv n$ elements is a permutation in which each element is labeled by a sign of plus or minus. A *segment* of π is a sequence of consecutive elements $\pi_i, \pi_{i+1}, \dots, \pi_k$, where $1 \leq i \leq k \leq n$. A *reversal* ρ is an operation that reverses the order of the elements in a segment and also flips their signs. Two segments $\pi_i, \pi_{i+1}, \dots, \pi_k$ and $\pi_j, \pi_{j+1}, \dots, \pi_l$ are said to be *contiguous* if $j = k + 1$ or $i = l + 1$. A *transposition* τ is an operation that exchanges two contiguous (disjoint) segments. A *transreversal* $\tau\rho_{A,B}$ (respectively, $\tau\rho_{B,A}$) is a transposition that exchanges two segments A and B and also reverses A (respectively, B). A *revrev* operation $\rho\rho$ reverses each of the two contiguous segments (without transposing them). The problem of finding a shortest sequence of transposition, transreversal and revrev operations that transforms a permutation into the identity permutation is called *sorting by transpositions, transreversals and revrevs*. The *distance* of a permutation π, denoted by $d(\pi)$, is the length of the shortest sorting sequence.

Key Results

Linear vs. Circular Permutations

An operation is said to *operate* on the affected segments as well as on the elements in those segments. Two operations μ and μ' are *equivalent* if they have the same rearrangement result, i. e., $\mu \cdot \pi = \mu' \cdot \pi$ for all π. In this work [8], Hartman and Sharan showed that for an element x of a circular permutation π, if μ is an operation that operates on x, then there exists an equivalent oper-



Sorting by Transpositions and Reversals (Approximate Ratio 1.5), Figure 1
a The equivalence of transreversal and revrev on circular permutations. b The breakpoint graph $G(\pi)$ of the permutation $\pi = [1, -4, 6, -5, 2, -7, -3]$, for which $f(\pi) = [1, 2, 8, 7, 11, 12, 10, 9, 3, 4, 14, 13, 6, 5]$. It is convenient to draw $G(\pi)$ on a circle such that *black edges* (i. e., *thick lines*) are on the circumference and *gray edges* (i. e., *thin lines*) are chords

ation μ' that does not operate on x. Based on this property, they further proved that the problem of sorting by transpositions, transreversals and revrevs is equivalent for linear and circular permutations. Moreover, they observed that revrevs and transreversals are equivalent operations for circular permutations (as illustrated in Fig. 1a), implying that the problem of sorting a linear/circular permutation by transpositions, transreversals and revrevs can be reduced to that of sorting a circular permutation by transpositions and transreversals only.

The Breakpoint Graph

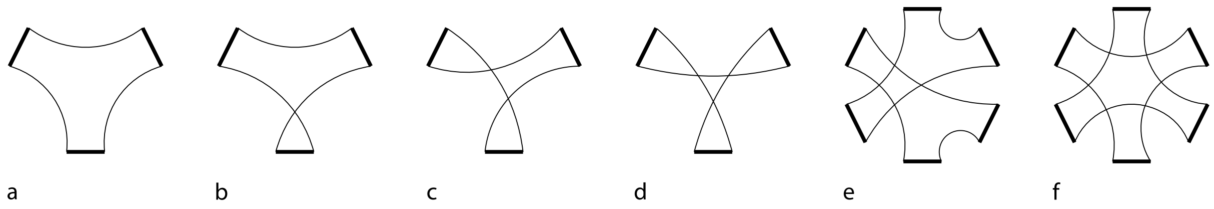
Given a signed permutation π on $\{1, 2, \dots, n\}$ of n elements, it is transformed into an unsigned permutation $f(\pi) = \pi' = [\pi'_1, \pi'_2, \dots, \pi'_{2n}]$ on $\{1, 2, \dots, 2n\}$ of $2n$ elements by replacing each positive element i with two elements $2i - 1, 2i$ (in this order) and each negative element $-i$ with $2i, 2i - 1$. The extended $f(\pi)$ is considered here as a circular permutation by identifying $2n + 1$ and 1 in both indices and elements. To ensure that every operation on $f(\pi)$ can be mimicked by an operation on π, only operations that cut before odd position are allowed for $f(\pi)$. The *breakpoint graph* $G(\pi)$ is an edge-colored graph on $2n$ vertices $\{1, 2, \dots, 2n\}$, in which for every $1 \le i \le n$, π'_{2i} is joined to π'_{2i+1} by a black edge and $2i$ is joined to $2i + 1$ by a gray edge (see Fig. 1b for an example). Since the degree of each vertex in $G(\pi)$ is exactly 2, $G(\pi)$ uniquely decomposes into cycles. A *k-cycle* (i. e., a cycle of *length k*) is a cycle with k black edges, and it is *odd* if k is odd. The number of odd cycles in $G(\pi)$ is denoted by $c_{\text{odd}}(\pi)$. It is not hard to verify that $G(\pi)$ consists of n 1-cycles and hence $c_{\text{odd}}(\pi) = n$, if π is an identity permutation $[1, 2, \dots, n]$. Gu et al. [5] have shown that $c_{\text{odd}}(\mu \cdot \pi) \le c_{\text{odd}}(\pi) + 2$ for all linear permutations π and operations μ. In this work [8], Hartman and Sharan further noted that the above result holds also for circular permutations and proved that the lower bound of $d(\pi)$ is $(n(\pi) - c_{\text{odd}}(\pi))/2$.

Transformation into 3-Permutations

A permutation is called *simple* if its breakpoint graph contains only k-cycle, where $k \le 3$. A simple permutation is also called a *3-permutation* if it contains no 2-cycles. A transformation from π to $\hat{\pi}$ is said to be *safe* if $n(\pi) - c_{\text{odd}}(\pi) = n(\hat{\pi}) - c_{\text{odd}}(\hat{\pi})$. It has been shown that every permutation π can be transformed into a simple one π' by safe transformations and, moreover, every sorting of π' mimics a sorting of π with the same number of operations [6,11]. Here, Hartman and Sharan [8] further showed that every simple permutation π' can be transformed into a 3-permutation $\hat{\pi}$ by safe paddings (of transforming those 2-cycles into 1-twisted 3-cycles) and, moreover, every sorting of $\hat{\pi}$ mimics a sorting of π' with the same number of operations. Hence, based on these two properties, an arbitrary permutation π can be transformed into a 3-permutation $\hat{\pi}$ such that every sorting of $\hat{\pi}$ mimics a sorting of π with the same number of operations, suggesting that one can restrict attention to circular 3-permutations only.

Cycle Types

An operation that cuts some black edges is said to *act* on these edges. An operation is further called a *k-operation* if it increases the number of odd cycles by k. A $(0, 2, 2)$-*sequence* is a sequence of three operations, of which the first is a 0-operation and the next two are 2-operations. An odd cycle is called *oriented* if there is a 2-operation that acts on three of its black edges; otherwise, it is *unori-*


Sorting by Transpositions and Reversals (Approximate Ratio 1.5), Figure 2
Configurations of 3-cycles. a Unoriented, 0-twisted 3-cycle. b Unoriented, 1-twisted 3-cycle. c Oriented, 2-twisted 3-cycle. d Oriented, 3-twisted 3-cycle. e A pair of intersecting 3-cycles. f A pair of interleaving 3-cycles

ented. A *configuration* of cycles is a subgraph of the breakpoint graph that contains one ore more cycles. As shown in Fig. 2a–d, there are four possible configurations of single 3-cycles. A black edge is called *twisted* if its two adjacent gray edges cross each other in the circular breakpoint graph. A cycle is k-twisted if k of its black edges are twisted. For example, the 3-cycles in Fig. 2a–d are 0-, 1-, 2- and 3-twisted, respectively. Hartman and Sharan observed that a 3-cycle is oriented if and only if it is 2- or 3-twisted.

Cycle Configurations

Two pairs of black edges are called *intersecting* if they alternate in the order of their occurrence along the circle. A pair of black edges *intersects* with cycle C, if it intersects with a pair of black edges that belong to C. Cycles C and D *intersect* if there is a pair of black edges in C that intersects with D (see Fig. 2e). Two intersecting cycles are called *interleaving* if their black edges alternate in their order of occurrence along the circle (see Fig. 2f). Clearly, the relation between two cycles is one of (1) non-intersecting, (2) intersecting but non-interleaving and (3) interleaving. A pair of black edges is *coupled* if they are connected by a gray edge and when reading the edges along the cycle, they are read in the same direction. For example, all pairs of black edges in Fig. 2a are coupled. Gu et al. [5] have shown that given a pair of coupled black edges (b_1, b_2), there exists a cycle C that intersects with (b_1, b_2). A *1-twisted pair* is a pair of 1-twisted cycles, whose twists are consecutive on the circle in a configuration that consists of these two cycles only. A 1-twisted cycle is called *closed* in a configuration if its two coupled edges intersect with some other cycle in the configuration. A configuration is *closed* if at least one of its 1-twisted cycles is closed; otherwise, it is called *open*.

The Algorithm

The basic ideas of the Hartman and Sharan's 1.5-approximation algorithm [8] for the problem of sorting by transpositions, transreversals and revrevs are as follows. Hartman and Sharan reduced the problem to that of sorting a circular 3-permutation by transpositions and transreversals only and then focused on transforming the 3-cycles into 1-cycles in the breakpoint graph of this 3-permutation. By definition, an oriented (i. e., 2- or 3-twisted) 3-cycle admits a 2-operation and, therefore, they continued to consider unoriented (i. e., 0- or 1-twisted) 3-cycles only. Since configurations involving only 0-twisted 3-cycles were handled with $(0, 2, 2)$-sequences in [7], Hartman and Sharan restricted their attention to those configurations that consist of 0- and 1-twisted 3-cycles. They showed that these configurations are all closed and that it can be sorted by a $(0, 2, 2)$-sequence of operations for each of the following five possible closed configurations: (1) a closed configuration with two unoriented, interleaving 3-cycles that do not form a 1-twisted pair, (2) a closed configuration with two intersecting, 0-twisted 3-cycles, (3) a closed configuration with two intersecting, 1-twisted 3-cycles, (4) a closed configuration with a 0-twisted 3-cycles that intersects with the coupled edges of a 1-twisted 3-cycle, and (5) a closed configuration that contains $k \geq 2$ mutually interleaving 1-twisted 3-cycles such that all their twists are consecutive on the circle and k is maximal with this property. As a result, the sequence of operations used by Hartman and Sharan in their algorithm contains only 2-operations and $(0, 2, 2)$-sequences. Since every sequence of three operations increases the number of odd cycles by at least 4 out of 6 possible in 3 steps, the ratio of their approximation algorithm is 1.5. Furthermore, Hartman and Sharan showed that their algorithm can be implemented in $O(n^{3/2}\sqrt{\log n})$ time using the data structure of Kaplan and Verbin [10], where n is the number of elements in the permutation.

Theorem 1 *The problem of sorting linear permutations by transpositions, transreversals and revrevs is linearly equiv-*

alent to the problem of sorting circular permutations by transpositions, transreversals and revrevs.

Theorem 2 *There is a 1.5-approximation algorithm for sorting by transpositions, transreversals and revrevs, which runs in $O(n^{3/2}\sqrt{\log n})$ time.*

Applications

When trying to determine evolutionary distance between two organisms using genomic data, biologists may wish to reconstruct the sequence of evolutionary events that have occurred to transform one genome into the other. One of the most promising ways to do this phylogenetic study is to compare the order of appearance of identical (e. g., orthologous) genes in two different genomes [9,12]. This comparison of computing global rearrangement events (such as reversals, transpositions and transreversals of genome segments) may provide more accurate and robust clues to the evolutionary process than the analysis of local point mutations (i. e., substitutions, insertions and deletions of nucleotides/amino acids). Usually, the two genomes being compared are represented by signed permutations, with each element standing for a gene and its sign representing the (transcriptional) direction of the corresponding gene on a chromosome. Then the goal of the resulting genome rearrangement problem is to find a shortest sequence of rearrangement operations that transforms (or, equivalently, *sorts*) one permutation into the other. Previous work focused on the problem of sorting a permutation by reversals. This problem has been shown by Capara [2] to be NP-hard, if the considered permutation is unsigned. However, for signed permutations, this problem becomes tractable and Hannenhalli and Pevzer [6] gave the first polynomial-time algorithm for it. On the other hand, there has been less progress on the problem of sorting by transpositions. Thus far, the complexity of this problem is still open, although several 1.5-approximation algorithms [1,3,7] have been proposed for it. Recently, the approximation ratio of sorting by transpositions was further improved to 1.375 by Elias and Hartman [4]. Gu et al. [5] and Lin and Xue [11] gave quadratic-time 2-approximation algorithms for sorting signed, linear permutations by transpositions and transreversals. In [11], Lin and Xue considered the problem of sorting signed, linear permutations by transpositions, transreversals and revrevs, and proposed a quadratic-time 1.75-approximation algorithm for it. In this work [8], Hartman and Sharan further showed that this problem is equivalent for linear and circular permutations and can be reduced to that of sorting signed, circular permutations by transpositions and transreversals only. In addition, they provided a 1.5-approximation algorithm that can be implemented in $O(n^{3/2}\sqrt{\log n})$ time.

Cross References

► Sorting Signed Permutations by Reversal (Reversal Distance)
► Sorting Signed Permutations by Reversal (Reversal Sequence)

Recommended Reading

1. Bafna, V., Pevzner, P.A.: Sorting by transpositions. SIAM J. Discret. Math. **11**, 224–240 (1998)
2. Caprara, A.: Sorting permutations by reversals and Eulerian cycle decompositions. SIAM J. Discret. Math. **12**, 91–110 (1999)
3. Christie, D.A.: Genome Rearrangement Problems. Ph. D. thesis, Department of Computer Science. University of Glasgow, U.K. (1999)
4. Elias, I., Hartman, T.: A 1.375-approximation algorithm for sorting by transpositions. IEEE/ACM Transactions on Computational Biology and Bioinformatics **3**, 369–379 (2006)
5. Gu, Q.P., Peng, S., Sudborough, H.: A 2-approximation algorithm for genome rearrangements by reversals and transpositions. Theor. Comput. Sci. **210**, 327–339 (1999)
6. Hannenhalli, S., Pevzner, P.A.: Transforming cabbage into turnip: polynomial algorithm for sorting signed permutations by reversals. J. Assoc. Comput. Mach. **46**, 1–27 (1999)
7. Hartman, T., Shamir, R.: A simpler and faster 1.5-approximation algorithm for sorting by transpositions. Inf. Comput. **204**, 275–290 (2006)
8. Hartman, T., Sharan, R.: A 1.5-approximation algorithm for sorting by transpositions and transreversals. In: Proceedings of the 4th Workshop on Algorithms in Bioinformatics (WABI'04), pp. 50–61. Bergen, Norway, 17–21 Sep (2004)
9. Hoot, S.B., Palmer, J.D.: Structural rearrangements, including parallel inversions, within the chloroplast genome of Anemone and related genera. J. Mol. Evol. **38**, 274–281 (1994)
10. Kaplan, H., Verbin, E.: Efficient data structures and a new randomized approach for sorting signed permutations by reversals. In: Proceedings of the 14th Annual Symposium on Combinatorial Pattern Matching (CPM'03), pp. 170–185. Morelia, Michocán, Mexico, 25–27 Jun (2003)
11. Lin, G.H., Xue, G.: Signed genome rearrangements by reversals and transpositions: models and approximations. Theor. Comput. Sci. **259**, 513–531 (2001)
12. Palmer, J.D., Herbon, L.A.: Tricircular mitochondrial genomes of Brassica and Raphanus: reversal of repeat configurations by inversion. Nucleic Acids Res. **14**, 9755–9764 (1986)

Spanning Ratio

► Algorithms for Spanners in Weighted Graphs
► Dilation of Geometric Networks
► Geometric Dilation of Geometric Networks

Sparse Graph Spanners

2004; Elkin, Peleg

MICHAEL ELKIN
Department of Computer Science,
Ben-Gurion University, Beer-Sheva, Israel

Keywords and Synonyms

$(1+\epsilon, \beta)$-spanners; Almost additive spanners

Problem Definition

For a pair of numbers α, β, $\alpha \geq 1$, $\beta \geq 0$, a subgraph $G' = (V, H)$ of an unweighted undirected graph $G = (V, E)$, $H \subseteq E$, is an (α, β)-spanner of G if for every pair of vertices $u, w \in V$, $\text{dist}_{G'}(u, w) \leq \alpha \cdot \text{dist}_G(u, w) + \beta$, where $\text{dist}_G(u, w)$ stands for the distance between u and w in G. It is desirable to show that for every n-vertex graph there exists a sparse (α, β)-spanner with as small values of α and β as possible. The problem is to determine asymptotic tradeoffs between α and β on one hand, and the sparsity of the spanner on the other.

Key Results

The main result of Elkin and Peleg [6] establishes the existence and efficient constructibility of $(1+\epsilon, \beta)$-spanners of size $O(\beta n^{1+1/\kappa})$ for every n-vertex graph G, where $\beta = \beta(\epsilon, \kappa)$ is constant whenever κ and ϵ are. The specific dependence of β on κ and ϵ is $\beta(\kappa, \epsilon) = \kappa^{\log\log\kappa - \log\epsilon}$.

An important ingredient of the construction of [6] is a partition of the graph G into regions of small diameter in such a way that the super-graph induced by these regions is sparse. The study of such partitions was initiated by Awerbuch [2], that used them for network synchronization. Peleg and Schäffer [8] were the first to employ such partitions for constructing spanners. Specifically, they constructed $(O(\kappa), 1)$-spanners with $O(n^{1+1/\kappa})$ edges. Althofer et al. [1] provided an alternative proof of the result of Peleg and Schäffer that uses an elegant greedy argument. This argument also enabled Althofer et al. to extend the result to weighted graphs, to improve the constant hidden by the O-notation in the result of Peleg and Schäffer, and to obtain related results for planar graphs.

Applications

Efficient algorithms for computing sparse $(1+\epsilon, \beta)$-spanners were devised in [5] and [11]. The algorithm of [5] was used in [5,7,10] for computing almost shortest paths in centralized, distributed, streaming, and dynamic centralized models of computations. The basic approach used in these results is to construct a sparse spanner, and then to compute exact shortest paths on the constructed spanner. The sparsity of the latter guarantees that the computation of shortest paths in the spanner is far more efficient than in the original graph.

Open Problems

The main open question is whether it is possible to achieve similar results with $\epsilon = 0$. More formally, the question is: Is it true that for any $\kappa \geq 1$ and any n-vertex graph G there exists $(1, \beta(\kappa))$-spanner of G with $O(n^{1+1/\kappa})$ edges? This question was answered in affirmitive for κ equal to 2 and 3 [3,4,6]. Some lower bounds were recently proved by Woodruff [12].

A less challenging problem is to improve the dependence of β on ϵ and κ. Some progress in this direction was achieved by Thorup and Zwick [11], and very recently by Pettie [9].

Cross References

► Synchronizers, Spanners

Recommended Reading

1. Althofer, I., Das, G., Dobkin, D.P., Joseph, D., Soares, J.: On Sparse Spanners of Weighted Graphs. Discret. Comput. Geom. **9**, 81–100 (1993)
2. Awerbuch, B.: Complexity of network synchronization. J. ACM **4**, 804–823 (1985)
3. Baswana, S., Kavitha, T., Mehlhorn, K., Pettie, S.: New Constructions of (alpha, beta)-spanners and purely additive spanners. In: Proc. of Symp. on Discrete Algorithms, Vancouver, Jan 2005, pp. 672–681
4. Dor, D., Halperin, S., Zwick, U.: All Pairs Almost Shortest Paths. SIAM J. Comput. **29**, 1740–1759 (2000)
5. Elkin, M.: Computing Almost Shortest Paths. Trans. Algorithms **1**(2), 283–323 (2005)
6. Elkin, M., Peleg, D.: $(1+\epsilon, \beta)$-Spanner Constructions for General Graphs. SIAM J. Comput. **33**(3), 608–631 (2004)
7. Elkin, M., Zhang, J.: Efficient Algorithms for Constructing $(1+\epsilon, \beta)$-spanners in the Distributed and Streaming Models. Distrib. Comput. **18**(5), 375–385 (2006)
8. Peleg, D., Schäffer, A.: Graph spanners. J. Graph Theory **13**, 99–116 (1989)
9. Pettie, S.: Low-Distortion Spanners. In: 34th International Colloquium on Automata Languages and Programm, Wroclaw, July 2007, pp. 78–89
10. Roditty, L., Zwick, U.: Dynamic approximate all-pairs shortest paths in undirected graphs. In: Proc. of Symp. on Foundations of Computer Science, Rome, Oct. 2004, pp. 499–508
11. Thorup, M., Zwick, U.: Spanners and Emulators with sublinear distance errors. In: Proc. of Symp. on Discrete Algorithms, Miami, Jan. 2006, pp. 802–809

12. Woodruff, D.: Lower Bounds for Additive Spanners, Emulators, and More. In: Proc. of Symp. on Foundations of Computer Science, Berckeley, Oct. 2006, pp. 389–398

Sparsest Cut

2004; Arora, Rao, Vazirani

SHUCHI CHAWLA
Department of Computer Science, University of Wisconsin–Madison, Madison, WI, USA

Keywords and Synonyms

Minimum ratio cut

Problem Definition

In the Sparsest Cut problem, informally, the goal is to partition a given graph into two or more large pieces while removing as few edges as possible. Graph partitioning problems such as this one occupy a central place in the theory of network flow, geometric embeddings, and Markov chains, and form a crucial component of divide-and-conquer approaches in applications such as packet routing, VLSI layout, and clustering.

Formally, given a graph $G = (V, E)$, the *sparsity* or *edge expansion* of a non-empty set $S \subset V$, $|S| \le \frac{1}{2}|V|$, is defined as follows:

$$\alpha(S) = \frac{|E(S, V \setminus S)|}{|S|} .$$

The sparsity of the graph, $\alpha(G)$, is then defined as follows:

$$\alpha(G) = \min_{S \subset V, |S| \le \frac{1}{2}|V|} \alpha(S) .$$

The goal in the Sparsest Cut problem is to find a subset $S \subset V$ with the minimum sparsity, and to determine the sparsity of the graph.

The first approximation algorithm for the Sparsest Cut problem was developed by Leighton and Rao in 1988 [13]. Employing a linear programming relaxation of the problem, they obtained an $O(\log n)$ approximation, where n is the size of the input graph. Subsequently Arora, Rao and Vazirani [4] obtained an improvement over Leighton and Rao's algorithm using a semi-definite programming relaxation, approximating the problem to within an $O(\sqrt{\log n})$ factor.

In addition to the Sparsest Cut problem, Arora et al. also consider the closely related Balanced Separator problem. A partition $(S, V \setminus S)$ of the graph G is called a c-balanced separator for $0 < c \le \frac{1}{2}$, if both S and $V \setminus S$ have at least $c|V|$ vertices. The goal in the Balanced Separator problem is to find a c-balanced partition with the minimum sparsity. This sparsity is denoted $\alpha_c(G)$.

Key Results

Arora et al. provide an $O(\sqrt{\log n})$ *pseudo-approximation* to the balanced-separator problem using semi-definite programming. In particular, given a constant $c \in (0, \frac{1}{2}]$, they produce a separator with balance c' that is slightly worse than c (that is, $c' < c$), but sparsity within an $O(\sqrt{\log n})$ factor of the sparsity of the optimal c-balanced separator.

Theorem 1 *Given a graph $G = (V, E)$, let $\alpha_c(G)$ be the minimum edge expansion of a c-balanced separator in this graph. Then for every fixed constant $a < 1$, there exists a polynomial-time algorithm for finding a c'-balanced separator in G, with $c' \ge ac$, that has edge expansion at most $O(\sqrt{\log n}\alpha_c(G))$.*

Extending this theorem to include unbalanced partitions, Arora et al. obtain the following:

Theorem 2 *Let $G = (V, E)$ be a graph with sparsity $\alpha(G)$. Then there exists a polynomial-time algorithm for finding a partition $(S, V \setminus S)$, with $S \subset V$, $S \ne \emptyset$, having sparsity at most $O(\sqrt{\log n}\alpha(G))$.*

An important contribution of Arora et al. is a new geometric characterization of vectors in n-dimensional space endowed with the squared-Euclidean metric. This result is of independent significance and has lead to or inspired improved approximation factors for several other partitioning problems (see, for example, [1,5,6,7,11]).

Informally, the result says that if a set of points in n-dimensional space is randomly projected on to a line, a good separator on the line is, with high probability, a good separator (in terms of squared-Euclidean distance) in the original high-dimensional space. Separation on the line is related to separation in the original space via the following definition of stretch.

Definition 1 (Def. 4 in [4]) Let $\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_n$ be a set of n points in $\mathcal{R}^n$, equipped with the squared-Euclidean metric $\mathrm{d}(x, y) = \|x - y\|_2^2$. The set of points is said to be *(t, γ, β)-stretched at scale ℓ*, if for at least a γ fraction of all the n-dimensional unit vectors u, there is a partial matching $M_u = \{(x_i, y_i)\}_i$ among these points, with $|M_u| \ge \beta n$, such that for all $(x, y) \in M_u$, $\mathrm{d}(x, y) \le \ell^2$ and $\langle u, \vec{x} - \vec{y} \rangle \ge t\ell/\sqrt{n}$. Here $\langle \cdot, \cdot \rangle$ denotes the dot product of two vectors.

Theorem 3 *For any $\gamma, \beta > 0$, there is a constant*

$C = C(\gamma, \beta)$ such that if $t > C \log^{1/3} n$, then no set of n points in $\mathcal{R}^n$ can be (t, γ, β)-stretched for any scale ℓ.

In addition to the SDP-rounding algorithm, Arora et al. provide an alternate algorithm for finding approximate sparsest cuts, using the notion of expander flows. This result leads to fast (quadratic time) implementations of their approximation algorithm [3].

Applications

One of the main applications of balanced separators is in improving the performance of divide and conquer algorithms for a variety of optimization problems.

One example is the Minimum Cut Linear Arrangement problem. In this problem, the goal is to order the vertices of a given n vertex graph G from 1 through n in such a way that the capacity of the largest of the cuts $(\{1, 2, \cdots, i\}, \{i + 1, \cdots, n\})$, $i \in [1, n]$, is minimized. Given a ρ-approximation to the balanced separator problem, the following divide and conquer algorithm gives an $O(\rho \log n)$-approximation to the Minimum Cut Linear Arrangement problem: find a balanced separator in the graph, then recursively order the two parts, and concatenate the orderings. The approximation follows by noting that if the graph has a balanced separator with expansion $\alpha_c(G)$, only $O(\rho n \alpha_n(G))$ edges are cut at every level, and given that a balanced separator is found at every step, the number of levels of recursion is at most $O(\log n)$.

Similar approaches can be used for problems such as VLSI layout and Gaussian elimination. (See the survey by Shmoys [14] for more details on these topics.)

The Sparsest Cut problem is also closely related to the problem of embedding squared-Euclidean metrics into the Manhattan (ℓ_1) metric with low distortion. In particular, the integrality gap of Arora et al.'s semi-definite programming relaxation for Sparsest Cut (generalized to include weights on vertices and capacities on edges) is exactly equal to the worst-case distortion for embedding a squared-Euclidean metric into the Manhattan metric. Using the technology introduced by Arora et al., improved embeddings from the squared-Euclidean metric into the Manhattan metric have been obtained [5,7].

Open Problems

Hardness of approximation results for the Sparsest Cut problem are fairly weak. Recently Chuzhoy and Khanna [9] showed that this problem is APX-hard, that is, there exists a constant $\epsilon > 0$, such that a $(1 + \epsilon)$-approximation algorithm for Sparsest Cut would imply P=NP. It is conjectured that the weighted version of the problem is NP-hard to approximate better than $O((\log \log n)^c)$ for some constant c, but this is only known to hold true assuming a version of the so-called Unique Games conjecture [8,12]. On the other hand, the semi-definite programming relaxation of Arora et al. is known to have an integrality gap of $\Omega(\log \log n)$ even in the unweighted case [10]. Proving an unconditional super-constant hardness result for weighted or unweighted Sparsest Cut, or obtaining $o(\sqrt{\log n})$-approximations for these problems remain open.

The directed version of the Sparset Cut problem has also been studied, and is known to be hard to approximate within a $2^{\Omega(\log^{1-\epsilon} n)}$ factor [9]. On the other hand, the best approximation known for this problem only achieves a polynomial factor of approximation—a factor of $O(n^{11/23} \log^{O(1)} n)$ due to Aggarwal, Alon and Charikar [2].

Recommended Reading

1. Agarwal, A., Charikar, M., Makarychev, K., Makarychev, Y.: $O(\sqrt{\log n})$ approximation algorithms for Min UnCut, Min 2CNF Deletion, and directed cut problems. In: Proceedings of the 37th ACM Symposium on Theory of Computing (STOC), Baltimore, May 2005, pp. 573–581
2. Aggarwal, A., Alon, N., Charikar, M.: Improved approximations for directed cut problems. In: Proceedings of the 39th ACM Symposium on Theory of Computing (STOC), San Diego, June 2007, pp. 671–680
3. Arora, S., Hazan, E., Kale, S.: An $O(\sqrt{\log n})$ approximation to SPARSEST CUT in $\tilde{O}(n^2)$ time. In: Proceedings of the 45th IEEE Symposium on Foundations of Computer Science (FOCS), Rome, ITALY, 17–19 October 2004, pp. 238–247
4. Arora, S., Rao, S., Vazirani, U.: Expander Flows, Geometric Embeddings, and Graph Partitionings. In: Proceedings of the 36th ACM Symposium on Theory of Computing (STOC), Chicago, June 2004, pp. 222–231
5. Arora, S., Lee, J., Naor, A.: Euclidean Distortion and the Sparsest Cut. In: Proceedings of the 37th ACM Symposium on Theory of Computing (STOC), Baltimore, May 2005, pp. 553–562
6. Arora, S., Chlamtac, E., Charikar, M.: New approximation guarantees for chromatic number. In: Proceedings of the 38th ACM Symposium on Theory of Computing (STOC), Seattle, May 2006, pp. 215–224
7. Chawla, S., Gupta, A., Räcke, H.: Embeddings of Negative-type Metrics and An Improved Approximation to Generalized Sparsest Cut. In: Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA), Vancouver, January 2005, pp. 102–111
8. Chawla, S., Krauthgamer, R., Kumar, R., Rabani, Y., Sivakumar, D.: On the Hardness of Approximating Sparsest Cut and Multicut. In: Proceedings of the 20th IEEE Conference on Computational Complexity (CCC), San Jose, June 2005, pp. 144–153
9. Chuzhoy, J., Khanna, S.: Polynomial flow-cut gaps and hardness of directed cut problems. In: Proceedings of the 39th ACM Symposium on Theory of Computing (STOC), San Diego, June 2007 pp. 179–188

10. Devanur, N., Khot, S., Saket, R., Vishnoi, N.: Integrality gaps for Sparsest Cut and Minimum Linear Arrangement Problems. In: Proceedings of the 38th ACM Symposium on Theory of Computing (STOC), Seattel, May 2006, pp. 537–546
11. Feige, U., Hajiaghayi, M., Lee, J.: Improved approximation algorithms for minimum-weight vertex separators. In: Proceedings of the 37th ACM Symposium on Theory of Computing (STOC), Baltimore, May 2005, pp. 563–572
12. Khot, S., Vishnoi, N.: The Unique Games Conjecture, Integrality Gap for Cut Problems and the Embeddability of Negative-Type Metrics into ℓ_1. In: Proceedings of the 46th IEEE Symposium on Foundations of Computer Science (FOCS), Pittsburgh, October 2005, pp. 53–62
13. Leighton, F.T., Rao, S.B.: An Approximate Max-Flow Min-Cut Theorem for Uniform Multicommodity Flow Problems with Applications to Approximation Algorithms. In: Proceedings of the 29th IEEE Symposium on Foundations of Computer Science (FOCS), White Plains, October 1988, pp. 422–431
14. Shmoys, D.B.: Cut problems and their application to divide-and-conquer. In: Hochbaum, D.S. (ed.) Approximation Algorithms for NP-hard Problems, pp. 192–235. PWS Publishing, Boston (1997)

Spatial Databases and Search

► Quantum Algorithm for Search on Grids
► R-Trees

Speed Scaling

1995; Yao, Demers, Shenker

KIRK PRUHS
Department of Computer Science,
University of Pittsburgh, Pittsburgh, PA, USA

Keywords and Synonyms

Speed scaling; Voltage scaling; Frequency scaling

Problem Definition

Speed scaling is a power management technique in modern processor that allows the processor to run at different speeds. There is a power function $P(s)$ that specifies the power, which is energy used per unit of time, as a function of the speed. In CMOS-based processors, the cube-root rule states that $P(s) \approx s^3$. This is usually generalized to assume that $P(s) = s^\alpha$ form some constant α. The goals of power management are to reduce temperature and/or to save energy. Energy is power integrated over time. Theoretical investigations to date have assumed that there is a fixed ambient temperature and that the processor cools according to Newton's law, that is, the rate of cooling is proportional to the temperature difference between the processor and the environment.

In the resulting scheduling problems, the scheduler must not only have a job-selection policy to determine the job to run at each time, but also a speed scaling policy to determine the speed at which to run that job. The resulting problems are generally dual objective optimization problems. One objective is some quality of service measure for the schedule, and the other objective is temperature or energy.

We will consider problems where jobs arrive at the processor over time. Each job i has a release time r_i when it arrives at the processor, and a work requirement w_i. A job i run at speed s takes w_i/s units of time to complete.

Key Results

[5] initiated the theoretical algorithmic investigation of speed scaling problems. [5] assumed that each job i had a deadline d_i, and that the quality of service measure was deadline feasibility (each job completes by its deadline). [5] gives a greedy algorithm YDS to find the minimum energy feasible schedule. The job selection policy for YDS is to run the job with the earliest deadline. To understand the speed scaling policy for YDS, define the intensity of a time interval to be the work that must be completed in this time interval divided by the length of the time interval. YDS then finds the maximum intensity interval, runs the jobs that must be run in this interval at constant speed, eliminates these jobs and this time interval from the instance, and proceeds recursively. [5] gives two online algorithms: OA and AVR. In OA the speed scaling policy is the speed that YDS would run at, given the current state and given that no more jobs will be released in the future. In AVR, the rate at which each job is completed is constant between the time that a job is released and the deadline for that job. [5] showed that AVR is $2^{\alpha-1}\alpha^\alpha$-competitive with respect to energy.

The results in [5] were extended in [2]. [2] showed that OA is α^α-competitive with respect to energy. [2] proposed another online algorithm, BKP. BKP runs at the speed of the maximum intensity interval containing the current time, taking into account only the work that has been released by the current time. They show that the competitiveness of BKP with respect to energy is at most $2(\alpha/(\alpha-1))^\alpha e^\alpha$. They also show that BKP is e-competitive with respect to the maximum speed.

[2] initiated the theoretical algorithmic investigation of speed scaling to manage temperature. [2] showed that the deadline feasible schedule that minimizes maximum temperature can in principle be computed in poly-

nomial time. [2] showed that the competitiveness of BKP with respect to maximum temperature is at most $2^{\alpha+1} e^{\alpha}(6(\alpha/(\alpha-1))^{\alpha}+1)$.

[4] initiated the theoretical algorithmic investigation into speed scaling when the quality-of-service objective is average/total flow time. The flow time of a job is the delay from when a job is released until it is completed. [4] give a rather complicated polynomial-time algorithm to find the optimal flow time schedule for unit work jobs, given a bound on the energy available. It is easy to see that no $O(1)$-competitive algorithm exists for this problem.

[1] introduce the objective of minimizing a linear combination of energy used and total flow time. This has a natural interpretation if one imagines the user specifying how much energy he is willing to use to increase the flow time of a job by a unit amount. [1] give an $O(1)$-competitive online algorithm for the case of unit work jobs. [3] improves upon this result and gives a 4-competitive online algorithm. The speed scaling policies of the online algorithms in [1] and [3] essentially run as power equal to the number of unfinished jobs (in each case modified in a particular way to facilitate analysis of the algorithm). [3] extend these results to apply to jobs with arbitrary work, and even arbitrary weight. The speed scaling policy is essentially to run at power equal to the weight of the unfinished work. The expression for the resulting competitive ratio is a bit complicated but is approximately 8 when the cube-root rule holds.

The analysis of the online algorithms in [2] and [3] heavily relied on amortized local competitiveness. An online algorithm is locally competitive for a particular objective if for all times the rate of increase of that objective for the online algorithm, plus the rate of change of some potential function, is at most the competitive ratio times the rate of increase of the objective in any other schedule.

Applications

None

Open Problems

The outstanding open problem is probably to determine if there is an efficient algorithm to compute the optimal flow time schedule given a fixed energy bound.

Recommended Reading

1. Albers, S., Fujiwara, H.: Energy-efficient algorithms for flow time minimization. In: STACS. Lecture Notes in Computer Science, vol. 3884, pp. 621–633. Springer, Berlin (2006)
2. Bansal, N., Kimbrel, T., Pruhs, K.: Speed scaling to manage energy and temperature. J. ACM **54**(1) (2007)
3. Bansal, N., Pruhs, K., Stein, C.: Speed scaling for weighted flow. In: ACM/SIAM Symposium on Discrete Algorithms, 2007
4. Pruhs, K., Uthaisombut, P., Woeginger, G.: Getting the Best Response for Your Erg. In: Scandanavian Workshop on Algorithms and Theory, 2004
5. Yao, F., Demers, A., Shenker, S.: A scheduling model for reduced CPU energy. In: IEEE Syposium on Foundations of Computer Science, 1995, p. 374

Sphere Packing Problem

2001; Chen, Hu, Huang, Li, Xu

DANNY Z. CHEN
Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN, USA

Keywords and Synonyms

Ball packing; Disk packing

Problem Definition

The sphere packing problem seeks to pack spheres into a given geometric domain. The problem is an instance of geometric packing. Geometric packing is a venerable topic in mathematics. Various versions of geometric packing problems have been studied, depending on the shapes of packing domains, the types of packing objects, the position restrictions on the objects, the optimization criteria, the dimensions, etc. It also arises in numerous applied areas. The sphere packing problem under consideration here finds applications in radiation cancer treatment using Gamma Knife systems. Unfortunately, even very restricted versions of geometric packing problems (e. g., regular-shaped objects and domains in lower dimensional spaces) have been proved to be NP-hard. For example, for *congruent packing* (i. e., packing copies of the same object), it is known that the 2-D cases of packing fixed-sized congruent squares or disks in a simple polygon are NP-hard [7]. Baur and Fekete [2] considered a closely related *dispersion problem* of packing k congruent disks in a polygon of n vertices such that the radius of the disks is maximized; they proved that the dispersion problem cannot be approximated arbitrarily well in polynomial time unless P = NP, and gave a $\frac{2}{3}$-approximation algorithm for the L_{∞} disk case with a time bound of $O(n^{38})$.

Chen et al. [4] proposed a practically efficient heuristic scheme, called *pack-and-shake*, for the **congruent sphere packing** problem, based on computational geometry techniques. The problem is defined as follows.

The Congruent Sphere Packing Problem

Given a d-D polyhedral region $R(d = 2, 3)$ of n vertices and a value $r > 0$, find a packing SP of R using spheres of radius r, such that (i) each sphere is contained in R, (ii) no two distinct spheres intersect each other in their interior, and (iii) the ratio (called the packing density) of the covered volume in R by SP over the total volume of R is maximized.

In the above problem, one can view the spheres as "solid" objects. The region R is also called the *domain* or *container*. Without loss of generality, let $r = 1$.

Much work on congruent sphere packing studied the case of packing spheres into an unbounded domain or even the whole space [5]. There are also results on packing congruent spheres into a bounded region. Hochbaum and Maass [8] presented a unified and powerful *shifting technique* for designing pseudo-polynomial time approximation schemes for packing congruent squares into a rectilinear polygon. But, the high time complexities associated with the resulting algorithms restrict their applicability in practice. Another approach is to formulate a packing problem as a non-linear optimization problem, and resort to an available optimization software to generate packings; however, this approach works well only for small problem sizes and regular-shaped domains.

To reduce the running time yet achieve a dense packing, a common idea is to consider objects that form a certain lattice or double-lattice. A number of results were given on lattice packing of congruent objects in the whole (especially high dimensional) space [5]. For a bounded rectangular 2-D domain, Milenkovic [10] adopted a method that first finds the densest translational lattice packing for a set of polygonal objects in the whole plane, and then uses some heuristics to extract the actual bounded packing.

Key Results

The *pack-and-shake* scheme of Chen et al. [4] for packing congruent spheres in an irregular-shaped 2-D or 3-D bounded domain R consists of three phases. In the first phase, the d-D domain R is partitioned into a set of convex subregions (called *cells*). The resulting set of cells defines a dual graph G_D, such that each vertex v of G_D corresponds to a cell $C(v)$ and an edge connects two vertices if and only if their corresponding cells share a $(d-1)$-D face. In the second phase, the algorithm repeats the following *trimming and packing* process until $G_D = \emptyset$: Remove the lowest degree vertex v from G_D and pack the cell $C(v)$. In the third phase, a *shake* procedure is applied to globally adjust the packing to obtain a denser one.

The objective of the trimming and packing procedure is that after each cell is packed, the remaining "packable" subdomain R' of R is always kept as a connected region. The rationale for maintaining the connectivity of R' is as follows. To pack spheres in a bounded domain R, two typical approaches have been used: (a) packing spheres layer by layer going from the boundary of R towards its interior [9], and (b) packing spheres starting from the "center" of R, such as its medial axis, towards its boundary [3,13,14]. Due to the shape irregularity of R, both approaches may fragment the remaining "packable" subdomain R' into more and more disconnected regions; however, at the end of packing each such region, a small "unpackable" area may eventually remain that allows no further packing. It could fit more spheres if the "packable" subdomain R' is lumped together instead of being divided into fragments, which is what the trimming and packing procedure aims to achieve.

Due to the packing of its adjacent cells that have been done by the trimming and packing procedure, the boundary of a cell $C(v)$ that is to be packed may consist of both line segments and arcs (from packed spheres). Hence, a key problem is to pack spheres in a cell bounded by curves of low degrees. Chen et al.'s algorithms [4] for packing each cell are based on certain lattice structures and allow the cell to both translate and rotate. Their algorithms have fairly low time bounds. In certain cases, they even run in nearly linear time.

An interesting feature of the cell packings generated by the trimming and packing procedure is that the resulted spheres cluster together in the middle of the cells of the domain R, leaving some small unpackable areas scattered along the boundary of R. The "shake" procedure in [4] thus seeks to collect these small areas together by "pushing" the spheres towards the boundary of R, in the hope of obtaining some "packable" region in the middle of R.

The approach in [4] is to first obtain a densest lattice unit sphere packing $LSP(C)$ for each cell C of R, and then use a "shake" procedure to globally adjust the resulting packing of R to generate a denser packing SP in R. Suppose the plane P is already packed by infinitely many unit spheres whose center points form a lattice (e. g., the hexagonal lattice). To obtain a densest packing $LSP(C)$ for a cell C from the lattice packing of the plane P, a position and orientation of C on P need to be computed such that C contains the maximum number of spheres from the lattice packing of P. There are two types of algorithms in [4] for computing an optimal placement of C on P: translational algorithms that allow C to be translated only, and translational/rotational algorithms that allow C to be both translated and rotated.

Let $n = |C|$, the number of bounding curves of C, and m be the number of spheres along the boundary of C in a sought optimal packing of C.

Theorem 1 *Given a polygonal region C bounded by n algebraic curves of constant degrees, a densest lattice unit sphere packing of C based only on translational motion can be computed in $O(N \log N + K)$ time, where $N = f(n, m)$ is a function of n and m, and K is the number of intersections between N planar algebraic curves of constant degrees that are derived from the packing instance.*

Note: In the worst case, $N = f(n, m) = n \times m$. But in practice, N may be much smaller. The N planar algebraic curves in Theorem 1 form a structure called *arrangement*. Since all these curves are of a constant degree, any two such curves can intersect each other at most a constant number of times. In the worst case, the number K of intersections between the N algebraic curves, which is also the *size* of the arrangement, is $O(N^2)$. The arrangement of these curves can be computed by the algorithms [1,6] in $O(N \log N + K)$ time.

Theorem 2 *Given a polygonal region C bounded by n algebraic curves of constant degrees, a densest lattice unit sphere packing of C based on both translational and rotational motions can be computed in $O(T(n) + (N + K') \log N)$ time, where $N = f(n, m)$ is a function of n and m, K' is the size of the arrangement of N pseudo-plane surfaces in 3-D that are derived from the packing instance, and $T(n)$ is the time for solving $O(n^2)$ quadratic optimization problem instances associated with the packing instance.*

In Theorem 2, $K' = O(N^3)$ in the worst case. In practice, K' can be much smaller.

The results on 2-D sphere packing in [4] can be extended to d-D for any constant integer $d \geq 3$, so long as a good d-D lattice packing of the d-D space is available.

Applications

Recent interest in the considered congruent sphere packing problem was motivated by medical applications in Gamma Knife radiosurgery [4,11,12]. Radiosurgery is a minimally invasive surgical procedure that uses radiation to destroy tumors inside human body while sparing the normal tissues. The Gamma Knife is a radiosurgical system that consists of 201 Cobalt-60 sources [3,14]; the gamma-rays from these sources are all focused on a common center point, thus creating a spherical volume of radiation field. The Gamma Knife treatment normally applies high radiation dose. In this setting, overlapping spheres may result in overdose regions (called *hot spots*) in the target treatment domain, while a low packing density may cause underdose regions (called *cold spots*) and a non-uniform dose distribution. Hence, one may view the spheres used in Gamma Knife packing as "solid" spheres. Therefore, a key geometric problem in Gamma Knife treatment planning is to fit multiple spheres into a 3-D irregular-shaped tumor [3,13,14]. The total treatment time crucially depends on the number of spheres used. Subject to a given packing density, the minimum number of spheres used in the packing (i. e., treatment) is desired. The Gamma Knife currently produces spheres of four different radii (4 mm, 8 mm, 14 mm, and 18 mm), and hence the Gamma Knife sphere packing is in general not congruent. In practice, a commonly used approach is to pack larger spheres first, and then fit smaller spheres into the remaining subdomains, in the hope of reducing the total number of spheres involved and thus shortening the treatment time. Therefore, congruent sphere packing can be used as a key subroutine for such a common approach.

Open Problems

An open problem is to analyze the quality bounds of the resulting packing for the algorithms in [4]; such packing quality bounds are currently not yet known. Another open problem is to reduce the running time of the packing algorithms in [4], since these algorithms, especially for sphere packing problems in higher dimensions, are still very time-consuming. In general, it is highly desirable to develop efficient sphere packing algorithms in d-D ($d \geq 2$) with guaranteed good packing quality.

Experimental Results

Some experimental results of the 2-D pack-and-shake sphere packing algorithms were given in [4]. The planar hexagonal lattice was used for the lattice packing. On packings whose sizes are in the hundreds, the C++ programs of the algorithms in [4] based only on translational motion run very fast (a few minutes), while those of the algorithms based on both translation and rotation take much longer time (hours), reflecting their respective theoretical time bounds, as expected. On the other hand, the packing quality of the translation-and-rotation based algorithms is a little better than the translation based algorithms. The packing densities of all the algorithms in the experiments are well above 70% and some are even close to or above 80%. Comparing with the nonconvex programming methods, the packing algorithms in [4] seemed to run faster based on the experiments.

Cross References

► Local Approximation of Covering and Packing Problems

Recommended Reading

1. Amato, N.M., Goodrich, M.T., Ramos, E.A.: Computing the arrangement of curve segments: Divide-and-conquer algorithms via sampling. In: Proc. 11th Annual ACM-SIAM Symp. on Discrete Algorithms, pp. 705–706 (2000)
2. Baur, C., Fekete, S.P.: Approximation of geometric dispersion problems. Algorithmica **30**(3), 451–470 (2001)
3. Bourland, J.D., Wu, Q.R.: Use of shape for automated, optimized 3D radiosurgical treatment planning. SPIE Proc. Int. Symp. on Medical Imaging, pp. 553–558 (1996)
4. Chen, D.Z., Hu, X., Huang, Y., Li, Y., Xu, J.: Algorithms for congruent sphere packing and applications. Proc. 17th Annual ACM Symp. on Computational Geometry, pp. 212–221 (2001)
5. Conway, J.H., Sloane, N.J.A.: Sphere Packings, Lattices and Groups. Springer, New York (1988)
6. Edelsbrunner, H., Guibas, L.J., Pach, J., Pollack, R., Seidel, R., Sharir, M.: Arrangements of curves in the plane: Topology, combinatorics, and algorithms. Theor. Comput. Sci. **92**, 319–336 (1992)
7. Fowler, R.J., Paterson, M.S., Tanimoto, S.L.: Optimal packing and covering in the plane are NP-complete. Inf. Process. Lett. **12**(3), 133–137 (1981)
8. Hochbaum, D.S., Maass, W.: Approximation schemes for covering and packing problems in image processing and VLSI. J. ACM **32**(1), 130–136 (1985)
9. Li, X.Y., Teng, S.H., Üngör, A.: Biting: Advancing front meets sphere packing. Int. J. Num. Methods Eng. **49**(1–2), 61–81 (2000)
10. Milenkovic, V.J.: Densest translational lattice packing of non-convex polygons. Proc. 16th ACM Annual Symp. on Computational Geometry, 280–289 (2000)
11. Shepard, D.M., Ferris, M.C., Ove, R., Ma, L.: Inverse treatment planning for Gamma Knife radiosurgery. Med. Phys. **27**(12), 2748–2756 (2000)
12. Sutou, A., Dai, Y.: Global optimization approach to unequal sphere packing problems in 3D. J. Optim. Theor. Appl. **114**(3), 671–694 (2002)
13. Wang, J.: Medial axis and optimal locations for min-max sphere packing. J. Combin. Optim. **3**, 453–463 (1999)
14. Wu, Q.R.: Treatment planning optimization for Gamma unit radiosurgery. Ph. D. Thesis, The Mayo Graduate School (1996)

Squares and Repetitions

1999; Kolpakov, Kucherov

Maxime Crochemore[1,2], Wojciech Rytter[3]
[1] Department of Computer Science, King's College London, London, UK
[2] Laboratory of Computer Science, University of Paris-East, Paris, France
[3] Institute of Informatics, Warsaw University, Warsaw, Poland

Keywords and Synonyms

Powers; Runs; Tandem repeats

Problem Definition

Periodicities and repetitions in strings have been extensively studied and are important both in theory and practice (combinatorics of words, pattern-matching, computational biology). The words of the type ww and www, where w is a nonempty primitive (not of the form u^k for an integer $k > 1$) word, are called squares and cubes, respectively. They are well-investigated objects in combinatorics on words [16] and in string-matching with small memory [5].

A string w is said to be periodic iff $period(w) \le |w|/2$, where $period(w)$ is the smallest positive integer p for which $w[i] = w[i + p]$ whenever both sides of the equality are defined. In particular each square and cube is periodic.

A repetition in a string $x = x_1 x_2 \dots x_n$ is an interval $[i \mathinner{.\,.} j] \subseteq [1 \mathinner{.\,.} n]$ for which the associated factor $x[i \mathinner{.\,.} j]$ is periodic. It is an occurrence of a periodic word $x[i \mathinner{.\,.} j]$, also called a positioned repetition. A word can be associated with several repetitions, see Fig. 1.

Initially people investigated mostly positioned squares, but their number is $\Omega(n \log n)$ [2], hence algorithms computing all of them cannot run in linear time, due to the potential size of the output. The optimal algorithms reporting all positioned squares or just a single square were designed in [1,2,3,19]. Unlike this, it is known that only $O(n)$ (un-positioned) squares can appear in a string of length n [8].

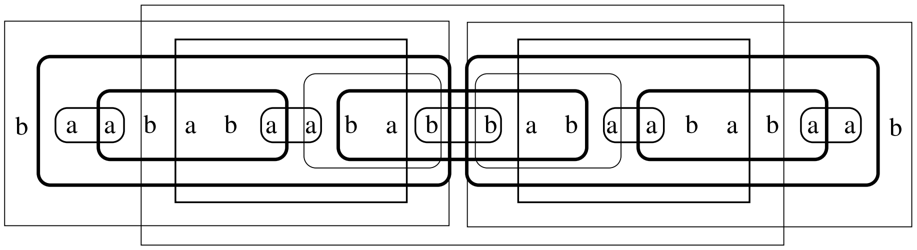
The concept of maximal repetitions, called runs (equivalent terminology) in [14], has been introduced to represent all repetitions in a succinct manner. The crucial property of runs is that there are only $O(n)$ runs in a word of length n [15,21].

A *run* in a string x is an interval $[i \mathinner{.\,.} j]$ such that both the associated string $x[i \mathinner{.\,.} j]$ has period $p \le (j - i + 1)/2$, and the periodicity cannot be extended to the right nor to the left: $x[i-1] \ne x[x + p - 1]$ and $x[j - p + 1] \ne x[j + 1]$ when the elements are defined. The set of runs of x is denoted by $RUNS(x)$. An example is displayed in Fig. 1.

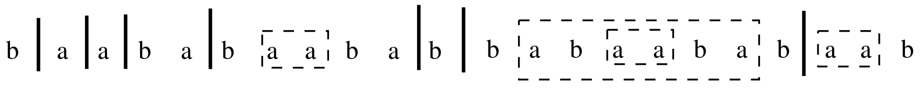
Key Results

The main results concern fast algorithms for computing positioned squares and runs, as well as combinatorial estimation on the number of corresponding objects.

Theorem 1 (Crochemore [1], Apostolico-Preparata [2], Main-Lorentz [19]) *There exists an $O(n \log n)$ worst-case*



Squares and Repetitions, Figure 1
The structure of *RUNS*(x) where x = baababaababbabaababaab = b$z^2(z^R)^2$b, for z = aabab. The operation $\cdot^R$ is reversing the string



Squares and Repetitions, Figure 2
The f-factorization of the example string x = baababaababbabaababaab and the set of its internal runs; all other runs overlap factorization points

time algorithm for computing all the occurrences of squares in a string of length n.

Techniques used to design the algorithms are based on partitioning, suffix trees, and naming segments. A similar result has been obtained by Franek, Smyth, and Tang using suffix arrays [11]. The key component in the next algorithm is the function described in the following lemma.

Lemma 2 (Main-Lorentz [19]) *Given two square-free strings u and v, reporting if uv contains a square centered in u can be done in worst-case time $O(|u|)$.*

Using suffix trees or suffix automata together with the function derived from the lemma, the following fact has been shown.

Theorem 3 (Crochemore [3], Main-Lorentz [19]) *Testing the square-freeness of a string of length n can be done in worst-case time $O(n \log a)$, where a is the size of the alphabet of the string.*

As a consequence of the algorithms and of the estimation on the number of squares, the most important result related to repetitions can be formulated as follows.

Theorem 4 (Kolpakov-Kucherov [15], Rytter [21], Crochemore-Ilie [4])
(1) All runs in a string can be computed in linear time (on a fixed-size alphabet).
(2) The number of all runs is linear in the length of the string.

The point (2) is very intricate, it is of purely combinatorial nature and has nothing to do with the algorithm. We sketch shortly the basic components in the constructive proof of the point (1). The main idea is to use, as for the previous theorem, the f-factorization (see [3]): a string x is decomposed into factors $u_1, u_2, \dots, u_k$, where u_i is the longest segment which appears before (possibly with overlap) or is a single letter if the segment is empty.

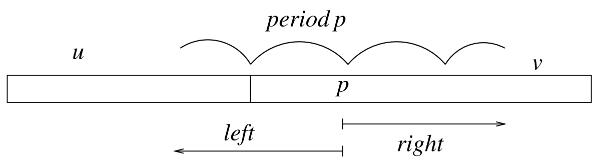
The runs which fit in a single factor are called internal runs, other runs are called here overlapping runs. There are three crucial facts:

- all overlapping runs can be computed in linear time,
- each internal run is a copy of an earlier overlapping run,
- the f-factorization can be computed in linear time (on a fixed-size alphabet) if we have the suffix tree or suffix automaton of the string. Figure 2 shows f-factorization and internal runs of an example string.

It follows easily from the definition of the f-factorization that if a run overlaps two (consecutive) factors u_{k-1} and u_k then its size is at most twice the total size of these two factors.

Figure 3 shows the basic idea for computing runs that overlap $u\,v$ in time $O(|u| + |v|)$. Using similar tables as in the Morris–Pratt algorithm (border and prefix tables), see [6], we can test the continuation of a period p from position p in v to the left and to the right. The corresponding tables can be constructed in linear time in a preprocessing phase. After computing all overlapping runs the internal runs can be copied from their earlier occurrences by processing the string from left to right.

Another interesting result concerning periodicities is the following lemma and its fairly immediate corollary.



Squares and Repetitions, Figure 3
If an overlapping run with period *p* starts in *u*, ends in *v*, and its part in *v* is of size at least *p* then it is easily detectable by computing continuations of the periodicity *p* in two directions: left and right

Lemma 5 (Three Prefix Squares, Crochemore-Rytter [5]) *If u, v, and w are three primitive words satisfying: $|u| < |v| < |w|$, uu is a prefix of vv, and vv is a prefix of ww, then $|u| + |v| \le |w|$*

Corollary 1 *Any nonempty string x possesses less than $\log_\Phi |y|$ prefixes that are squares.*

In the configuration of the lemma, a second consequence is that uu is a prefix of w. Therefore, a position in a string x cannot be the largest position of more than two squares, which yields the next corollary. A simple direct proof of it is by Ilie [13], see also [17].

Corollary 2 (Fraenkel and Simpson [8]) *Any string x contains at most $2|x|$ (different) squares, that is: $\text{card}\{u \mid u$ primitive and u^2 factor of $y\} \le 2|x|$.*

The structure of all squares and of un-positioned runs has been also computed within the same time complexities as above in [18] and [12].

Applications

Detecting repetitions in strings is an important element of several questions: pattern matching, text compression, and computational biology to quote a few. Pattern-matching algorithms have to cope with repetitions to be efficient as these are likely to slow down the process; the large family of dictionary-based text compression methods use a weaker notion of repeats (like the software gzip); repetitions in genomes, called satellites, are intensively studied because, for example, some over-repeated short segments are related to genetic diseases; some satellites are also used in forensic crime investigations.

Open Problems

The most intriguing question remains the asymptotically tight bound for the maximum number $\rho(n)$ of runs in a string of size n. The first proof (by painful induction) was quite difficult and has not produced any *concrete* constant coefficient in the $O(n)$ notation. This subject has been studied in [9,10,22,23]. The best-known lower bound of approximately $0.927\,n$ is from [10]. The exact number of runs has been considered for special strings: *Fibonacci words* and (more generally) *Sturmian words* [7,14,20]. It is proved in a structural and intricate manner in the full version of [21] that $\rho(n) \le 3.44\,n$, by introducing a *sparse-neighbors technique*. The neighbors are runs for which both the distance between their starting positions is small and the difference between their periods is also proportionally small (according to some fixed coefficient of proportionality). The occurrences of neighbors satisfy certain *sparsity* properties which imply the linear upper bound. Several variations for the definitions of neighbors and sparsity are possible. Considering runs having close centers the bound has been lowered to $1.6\,n$ in [4].

As a conclusion, we believe that the following fact is valid.

Conjecture: *A string of length n contains less than n runs, i. e., $|\text{RUNS}|(n) < n$.*

Cross References

Elements of the present entry are of main importance for run-length compression as well as for ▶ Run-length Compressed Pattern Matching. They are also related to the ▶ Approximate Tandem Repeats entries because "tandem repeat" is a synonym of repetition and "power."

Recommended Reading

1. Apostolico, A., Preparata, F.P.: Optimal off-line detection of repetitions in a string. Theor. Comput. Sci. **22**(3), 297–315 (1983)
2. Crochemore, M.: An optimal algorithm for computing the repetitions in a word. Inform. Process. Lett. **12**(5), 244–250 (1981)
3. Crochemore, M. : Transducers and repetitions. Theor. Comput. Sci. **45**(1), 63–86 (1986)
4. Crochemore, M., Ilie, L.: Analysis of maximal repetitions in strings. J. Comput. Sci. (2007)
5. Crochemore, M., Rytter, W.: Squares, cubes, and time-space efficient string searching. Algorithmica **13**(5), 405–425 (1995)
6. Crochemore, M., Rytter, W.: Jewels of stringology. World Scientific, Singapore (2003)
7. Franek, F., Karaman, A., Smyth, W.F.: Repetitions in Sturmian strings. Theor. Comput. Sci. **249**(2), 289–303 (2000)
8. Fraenkel, A.S., Simpson, R.J.: How many squares can a string contain? J. Comb. Theory Ser. A **82**, 112–120 (1998)
9. Fraenkel, A.S., Simpson, R.J.: The Exact Number of Squares in Fibonacci Words. Theor. Comput. Sci. **218**(1), 95–106 (1999)
10. Franek, F., Simpson, R.J. , and Smyth, W.F.: The maximum number of runs in a string. In: Proc. 14-th Australian Workshop on Combinatorial Algorithms, pp. 26–35. Curtin University Press, Perth (2003)

11. Franek, F., Smyth, W.F., Tang, Y.: Computing all repeats using suffix arrays. J. Autom. Lang. Comb. **8**(4), 579–591 (2003)
12. Gusfield, D.and Stoye, J.: Linear time algorithms for finding and representing all the tandem repeats in a string. J. Comput. Syst. Sci. **69**(4), 525–546 (2004)
13. Ilie, L.: A simple proof that a word of length n has at most 2n distinct squares. J. Combin. Theory, Ser. A **112**(1), 163–164 (2005)
14. Iliopoulos, C., Moore, D., Smyth, W.F.: A characterization of the squares in a Fibonacci string. Theor. Comput. Sci. **172** 281–291 (1997)
15. Kolpakov, R., Kucherov, G.: Finding maximal repetitions in a word in linear time. In: Proceedings of the 40th Symposium on Foundations of Computer Science, pp. 596–604. IEEE Computer Society Press, Los Alamitos (1999)
16. Lothaire, M. (ed.): Algebraic Combinatorics on Words. Cambridge University Press, Cambridge (2002)
17. Lothaire, M. (ed.): Applied Combinatorics on Words. Cambridge University Press, Cambridge (2005)
18. Main, M.G.: Detecting leftmost maximal periodicities. Discret. Appl. Math. **25**, 145–153 (1989)
19. Main, M.G., Lorentz, R.J.: An $O(n \log n)$ algorithm for finding all repetitions in a string. J. Algorithms **5**(3), 422–432 (1984)
20. Rytter, W.: The structure of subword graphs and suffix trees of Fibonacci words. In: Implementation and Application of Automata, CIAA 2005. Lecture Notes in Computer Science, vol. 3845, pp. 250–261. Springer, Berlin (2006)
21. Rytter, W.: The Number of Runs in a String: Improved Analysis of the Linear Upper Bound. In: Proceedings of the 23rd Annual Symposium on Theoretical Aspects of Computer Science. Lecture Notes in Computer Science, vol. 3884, pp. 184–195. Springer, Berlin (2006)
22. Smyth, W.F.: Repetitive perhaps, but certainly not boring. Theor. Comput. Sci. **249**(2), 343–355 (2000)
23. Smyth, W.F.: Computing patterns in strings. Addison-Wesley, Boston, MA (2003)

Stable Marriage

1962; Gale, Shapley

ROBERT W. IRVING
Department of Computing Science,
University of Glasgow, Glasgow, UK

Keywords and Synonyms

Stable matching

Problem Definition

The objective in *stable matching problems* is to match together pairs of elements of a set of participants, taking into account the preferences of those involved, and focusing on a stability requirement. The stability property ensures that no pair of participants would both prefer to be matched together rather than to accept their allocation in the matching. Such problems have widespread application, for example in the allocation of medical students to hospital posts, students to schools or colleges, etc.

An instance of the classical *Stable Marriage problem* (SM), introduced by Gale and Shapley [2], involves a set of $2n$ participants comprising n men $\{m_1, \dots, m_n\}$ and n women $\{w_1, \dots, w_n\}$. Associated with each participant is a *preference list*, which is a total order over the participants of the opposite sex. A man m_i *prefers* woman w_j to woman w_k if w_j precedes w_k on the preference list of m_i, and similarly for the women. A *matching* M is a bijection between the sets of men and women, in other words a set of man-woman pairs so that each man and each woman belongs to exactly one pair of M. For a man m_i, $M(m_i)$ denotes the *partner* of m_i in M, i. e., the unique woman w_j such that (m_i, w_j) is in M. Similarly, $M(w_j)$ denotes the partner of woman w_j in M. A matching M is *stable* if there is no *blocking pair*, namely a pair (m_i, w_j) such that m_i prefers w_j to $M(m_i)$ and w_j prefers m_i to $M(w_j)$.

Relaxing the requirements that the numbers of men and women are equal, and that each participant should rank *all* of the members of the opposite sex, gives the *Stable Marriage problem with Incomplete lists* (SMI). So an instance of SMI comprises a set of n_1 men $\{m_1, \dots, m_{n_1}\}$ and a set of n_2 women $\{w_1, \dots, w_{n_2}\}$, and each participant's preference list is a total order over a *subset* of the participants of the opposite sex. The implication is that if woman w_j does not appear on the list of man m_i then she is not an acceptable partner for m_i, and vice versa. A man-woman pair is *acceptable* if each member of the pair is on the preference list of the other, and a matching M is now a set of acceptable pairs such that each man and each woman is in *at most* one pair of M. In this context, a blocking pair for matching M is an acceptable pair (m_i, w_j) such that m_i is either unmatched in M or prefers w_j to $M(m_i)$, and likewise, w_j is either unmatched or prefers m_i to $M(w_j)$. A matching is stable if it has no blocking pair. So in an instance of SMI, a stable matching need not match all of the participants.

Gale and Shapley also introduced a many-one version of stable marriage, which they called the *College Admissions problem*, but which is now more usually referred to as the ▶ Hospitals/Residents Problem (HR) because of its well-known applications in the medical employment field. This problem is covered in detail in Entry 150 of this volume.

A comprehensive treatment of many aspects of the Stable Marriage problem, as of 1989, appears in the monograph of Gusfield and Irving [5].

Key Results

Theorem 1 *For every instance of SM or SMI there is at least one stable matching.*

Theorem 1 was proved constructively by Gale and Shapley [2] as a consequence of the algorithm that they gave to find a stable matching.

Theorem 2
(i) For a given instance of SM involving n men and n women, there is a $O(n^2)$ time algorithm that finds a stable matching.
(ii) For a given instance of SMI in which the combined lengths of all the preference lists is a, there is a $O(a)$ time algorithm that finds a stable matching.

The algorithm for SMI is a simple extension of that for SM. Each can be formulated in a variety of ways, but is most usually expressed in terms of a sequence of 'proposals' from the members of one sex to the members of the other. A pseudocode version of the SMI algorithm appears in Fig. 1, in which the traditional approach of allowing men to make proposals is adopted.

The complexity bound of Theorem 2(i) first appeared in Knuth's monograph on Stable Marriage [11]. The fact that this algorithm is asymptotically optimal was subsequently established by Ng and Hirschberg [15] via an adversary argument. On the other hand, Wilson [19] proved that the average running time, taken over all possible instances of SM, is $O(n \log n)$.

The algorithm of Fig. 1, in its various guises, has come to be known as the Gale–Shapley algorithm. The variant of the algorithm given here is called *man-oriented*, because men have the advantage of proposing. Reversing the roles of men and women gives the *woman-oriented* variant. The 'advantage' of proposing is remarkable, as spelled out in the next theorem.

Theorem 3 *The man-oriented version of the Gale–Shapley algorithm for SM or SMI yields the man-optimal stable matching in which each man has the best partner that he can have in any stable matching, but in which each woman has her worst possible partner. The woman-oriented version yields the woman-optimal stable matching, which has analogous properties favoring the women.*

The optimality property of Theorem 3 was established by Gale and Shapley [2], and the corresponding 'pessimality' property was first observed by McVitie and Wilson [14].

As observed earlier, a stable matching for an instance of SMI need not match all of the participants. But the following striking result was established by Gale and Sotomayor [3] and Roth [17] (in the context of the more general HR problem).

Theorem 4 *In an instance of SMI, all stable matchings have the same size and match exactly the same subsets of the men and women.*

For a given instance of SM or SMI, there may be many different stable matchings. Indeed Knuth [11] showed that the maximum possible number of stable matchings grows exponentially with the number of participants. He also pointed out that the set of stable matchings forms a distributive lattice under a natural dominance relation, a result attributed to Conway. This powerful algebraic structure that underlies the set of stable matchings can be exploited algorithmically in a number of ways. For example,

```
M = ∅;
assign each person to be free;        /* i. e., not a member of a pair in M */
while (some man m is free and has not proposed to every woman on his list)
    m proposes to w, the first woman on his list to whom he has not proposed;
    if (w is free)
        add (m, w) to M;              /* w accepts m */
    else if (w prefers m to her current partner m′)
        remove (m′, w) from M;        /* w rejects m′, setting m′ free */
        add (m, w) to M;              /* w accepts m */
    else
        M remains unchanged;          /* w rejects m */
return M;
```

Stable Marriage, Figure 1
The Gale–Shapley Algorithm

Gusfield [4] showed how all k stable matchings for an instance of SM can be generated in $O(n^2 + kn)$ time. ► Optimal Stable Marriage.

Extensions of these problems that are important in practice, so-called SMT and SMTI (extensions of SM and SMI respectively), allow the presence of *ties* in the preference lists. In this context, three different notions of stability have been defined [7] – *weak*, *strong* and *super-stability*, depending on whether the definition of a blocking pair requires that both members should improve, or at least one member improves and the other is no worse off, or merely that neither member is worse off. The following theorem summarizes the basic algorithmic results for these three varieties of stable matchings.

Theorem 5 *For a given instance of SMT or SMTI:*

(i) *A weakly stable matching is guaranteed to exist, and can be found in $O(n^2)$ or $O(a)$ time, respectively;*

(ii) *A super-stable matching may or may not exist; if one does exist it can be found in $O(n^2)$ or $O(a)$ time respectively;*

(iii) *A strongly stable matching may or may not exist; if one does exist it can be found in $O(n^3)$ or $O(na)$ time, respectively.*

Theorem 5 parts (i) and (ii) are due to Irving [7] (for SMT) and Manlove [12] (for SMTI). Part (iii) is due to Mehlhorn et al. [10], who improved earlier algorithms of Irving and Manlove.

It turns out that, in contrast to the situation described by Theorem 4(i), weakly stable matchings in SMTI can have different sizes. The natural problem of finding a maximum cardinality weakly stable matching, even under severe restrictions on the ties, is NP-hard [13]. ► Stable Marriage with Ties and Incomplete Lists explores this problem further.

The Stable Marriage problem is an example of a *bipartite* matching problem. The extension in which the bipartite requirement is dropped is the so-called *Stable Roommates* (SR) problem.

Gale and Shapley had observed that, unlike the case of SM, an instance of SR may or may not admit a stable matching, and Knuth [11] posed the problem of finding an efficient algorithm for SR, or proving it NP-complete. Irving [6] established the following theorem via a non-trivial extension of the Gale–Shapley algorithm.

Theorem 6 *For a given instance of SR, there exists a $O(n^2)$ time algorithm to determine whether a stable matching exists, and if so to find such a matching.*

Variants of SR may be defined, as for SM, in which preference lists may be incomplete and/or contain ties – these are denoted by SRI, SRT and SRTI – and in the presence of ties, the three flavors of stability, weak, strong and super, are again relevant.

Theorem 7 *For a given instance of SRT or SRTI:*

(i) *A weakly stable matching may or may not exist, and it is an NP-complete problem to determine whether such a matching exists;*

(ii) *A super-stable matching may or may not exist; if one does exist it can be found in $O(n^2)$ or $O(a)$ time respectively;*

(iii) *A strongly stable matching may or may not exist; if one does exist it can be found in $O(n^4)$ or $O(a^2)$ time, respectively.*

Theorem 7 part (i) is due to Ronn [16], part (ii) is due to Irving and Manlove [9], and part (iii) is due to Scott [18].

Applications

Undoubtedly the best known and most important applications of stable matching algorithms are in centralized matching schemes in the medical and educational domains. ► Hospitals/Residents Problem includes a summary of some of these applications.

Open Problems

The parallel complexity of stable marriage remains open. The best known parallel algorithm for SMI is due to Feder, Megiddo and Plotkin [1] and has $O(\sqrt{a}\log^3 a)$ running time using a polynomially bounded number of processors. It is not known whether the problem is in NC, but nor is there a proof of P-completeness.

One of the open problems posed by Knuth in his early monograph on stable marriage [11] was that of determining the maximum possible number x_n of stable matchings for any SM instance involving n men and n women. This problem remains open, although Knuth himself showed that x_n grows exponentially with n. Irving and Leather [8] conjecture that, when n is a power of 2, this function satisfies the recurrence

$$x_n = 3x_{n/2}^2 - 2x_{n/4}^4 .$$

Many open problems remain in the setting of weak stability, such as finding a good approximation algorithm for a maximum cardinality weakly stable matching – see ► Stable Marriage with Ties and Incomplete Lists – and enumerating all weakly stable matchings efficiently.

Cross References

▶ Hospitals/Residents Problem
▶ Optimal Stable Marriage
▶ Ranked Matching
▶ Stable Marriage and Discrete Convex Analysis
▶ Stable Marriage with Ties and Incomplete Lists
▶ Stable Partition Problem

Recommended Reading

1. Feder, T., Megiddo, N., Plotkin, S.A.: A sublinear parallel algorithm for stable matching. Theor. Comput. Sci. **233**(1–2), 297–308 (2000)
2. Gale, D., Shapley, L.S.: College admissions and the stability of marriage. Am. Math. Monthly **69**, 9–15 (1962)
3. Gale, D., Sotomayor, M.: Some remarks on the stable matching problem. Discret. Appl. Math. **11**, 223–232 (1985)
4. Gusfield, D.: Three fast algorithms for four problems in stable marriage. SIAM J. Comput. **16**(1), 111–128 (1987)
5. Gusfield, D., Irving, R.W.: The Stable Marriage Problem: Structure and Algorithms. MIT Press, Cambridge (1989)
6. Irving, R.W.: An efficient algorithm for the stable roommates problem. J. Algorithms **6**, 577–595 (1985)
7. Irving, R.W.: Stable marriage and indifference. Discret. Appl. Math. **48**, 261–272 (1994)
8. Irving, R.W., Leather, P.: The complexity of counting stable marriages. SIAM J. Comput. **15**(3), 655–667 (1986)
9. Irving, R.W., Manlove, D.F.: The stable roommates problem with ties. J. Algorithms **43**, 85–105 (2002)
10. Kavitha, T., Mehlhorn, K., Michail, D., Paluch, K.: Strongly stable matchings in time O(nm), and extension to the H/R problem. In: Proceedings of STACS 2004: the 21st Symposium on Theoretical Aspects of Computer Science. Lecture Notes in Computer Science, vol. 2996, pp. 222–233. Springer, Berlin (2004)
11. Knuth, D.E.: Mariages Stables. Les Presses de L'Université de Montréal, Montréal (1976)
12. Manlove, D.F.: Stable marriage with ties and unacceptable partners. Technical Report TR-1999-29, University of Glasgow, Department of Computing Science, January (1999)
13. Manlove, D.F., Irving, R.W., Iwama, K., Miyazaki, S., Morita, Y.: Hard variants of stable marriage. Theor. Comput. Sci. **276**(1–2), 261–279 (2002)
14. McVitie, D., Wilson, L.B.: The stable marriage problem. Commun. ACM **14**, 486–490 (1971)
15. Ng, C., Hirschberg, D.S.: Lower bounds for the stable marriage problem and its variants. SIAM J. Comput. **19**, 71–77 (1990)
16. Ronn, E.: NP-complete stable matching problems. J. Algorithms **11**, 285–304 (1990)
17. Roth, A.E.: The evolution of the labor market for medical interns and residents: a case study in game theory. J. Polit. Econ. **92**(6), 991–1016 (1984)
18. Scott, S.: A study of stable marriage problems with ties. Ph. D. thesis, University of Glasgow, Department of Computing Science (2005)
19. Wilson, L.B.: An analysis of the stable marriage assignment algorithm. BIT **12**, 569–575 (1972)

Stable Marriage and Discrete Convex Analysis

2000; Eguchi, Fujishige, Tamura, Fleiner

AKIHISA TAMURA
Department of Mathematics, Keio University, Yokohama, Japan

Keywords and Synonyms

Stable matching

Problem Definition

In the stable marriage problem first defined by Gale and Shapley [7], there are one set each of men and women having the same size, and each person has a strict preference order on persons of the opposite gender. The problem is to find a matching such that there is no pair of a man and a woman who prefer each other to their partners in the matching. Such a matching is called a *stable marriage* (or *stable matching*). Gale and Shapley showed the existence of a stable marriage and gave an algorithm for finding one. Fleiner [4] extended the stable marriage problem to the framework of matroids, and Eguchi, Fujishige, and Tamura [3] extended this formulation to a more general one in terms of discrete convex analysis, which was developed by Murota [8,9]. Their formulation is described as follows.

Let M and W be sets of men and women who attend a dance party at which each person dances a waltz T times and the number of times that he/she can dance with the same person of the opposite gender is unlimited. The problem is to find an "agreeable" allocation of dance partners, in which each person is assigned at most T persons of the opposite gender with possible repetition. Let $E = M \times W$, i. e., the set of all man-woman pairs. Also define $E_{(i)} = \{i\} \times W$ for all $i \in M$ and $E_{(j)} = M \times \{j\}$ for all $j \in W$. Denoting by $x(i, j)$ the number of dances between man i and woman j, an allocation of dance partners can be described by a vector $x = (x(i, j) : i \in M, j \in W) \in \mathbf{Z}^E$, where $\mathbf{Z}$ denotes the set of all integers. For each $y \in \mathbf{Z}^E$ and $k \in M \cup W$, denote by $y_{(k)}$ the restriction of y on $E_{(k)}$. For example, for an allocation $x \in \mathbf{Z}^E$, $x_{(k)}$ represents the allocation of person k with respect to x. Each person k describes his/her preferences on allocations by using a value function $f_k : \mathbf{Z}^{E_{(k)}} \to \mathbf{R} \cup \{-\infty\}$, where $\mathbf{R}$ denotes the set of all reals and $f_k(y) = -\infty$ means that allocation $y \in \mathbf{Z}^{E_{(k)}}$ is unacceptable for k. Note that the valuation of each person on allocations is determined only by his/her allocations. Let $\mathrm{dom}\, f_k = \{y \mid f_k(y) \in \mathbf{R}\}$. Assume that each value function f_k satisfies the following assumption:

(A) $\mathrm{dom}\, f_k$ is bounded and hereditary, and has $\mathbf{0}$ as the minimum point, where $\mathbf{0}$ is the vector of all zeros and heredity means that for any $y, y' \in \mathbf{Z}^{E(k)}, \mathbf{0} \le y' \le y \in \mathrm{dom}\, f_k$ implies $y' \in \mathrm{dom}\, f_k$.

For example, the following value functions with $M = \{1\}$ and $W = \{2, 3\}$

$$f_1(x(1,2), x(1,3)) = \begin{cases} 10(x(1,2)+x(1,3)) - x(1,2)^2 - x(1,3)^2 & \text{if } x(1,2), x(1,3) \ge 0 \\ & \text{and } x(1,2)+x(1,3) \le 3 \\ -\infty & \text{otherwise,} \end{cases}$$

$$f_j(x(1,j)) = \begin{cases} x(1,j) & \text{if } x(1,j) \in \{0,1,2,3\} (j = 2,3) \\ -\infty & \text{otherwise} \end{cases}$$

represent the case where (1) everyone wants to dance as many times, up to three, as possible, and (2) man 1 wants to divide his dances between women 2 and 3 as equally as possible. Allocations $(x(1,2), x(1,3)) = (1,2)$ and (2,1) are stable in the sense below.

A vector $x \in \mathbf{Z}^E$ is called a *feasible allocation* if $x_{(k)} \in \mathrm{dom}\, f_k$ for all $k \in M \cup W$. An allocation x is said to satisfy *incentive constraints* if each person has no incentive to unilaterally decrease the current units of x, that is if it satisfies

$$f_k(x_{(k)}) = \max\{f_k(y) \mid y \le x_{(k)}\} \qquad (\forall k \in M \cup W). \quad (1)$$

An allocation x is called *unstable* if it does not satisfy incentive constraints or there exist $i \in M$, $j \in W$, $y' \in \mathbf{Z}^{E(i)}$ and $y'' \in \mathbf{Z}^{E(j)}$ such that

$$f_i(x_{(i)}) < f_i(y'), \quad (2)$$

$$y'(i, j') \le x(i, j') \qquad (\forall j' \in W \setminus \{j\}), \quad (3)$$

$$f_j(x_{(j)}) < f_j(y''), \quad (4)$$

$$y''(i', j) \le x(i', j) \qquad (\forall i' \in M \setminus \{i\}), \quad (5)$$

$$y'(i, j) = y''(i, j). \quad (6)$$

Conditions (2) and (3) say that man i can strictly increase his valuation by changing the current number of dances with j without increasing the numbers of dances with other women, and (4) and (5) describe a similar situation for women. Condition (6) requires that i and j agree on the number of dances between them. An allocation x is called *stable* if it is not unstable.

Problem 1 *Given disjoint sets M and W, and value functions $f_k : \mathbf{Z}^{E(k)} \to \mathbf{R} \cup \{-\infty\}$ for $k \in M \cup W$ satisfying assumption* (A)*, find a stable allocation x.*

Remark 1 A time schedule for a given feasible allocation can be given by a famous result on graph coloring, namely, "any bipartite graph can be edge-colorable with the maximum degree colors."

Key Results

The work of Eguchi, Fujishige, and Tamura [3] gave a solution to Problem 1 in the case where each value function f_k is M$^\natural$-concave.

Discrete Convex Analysis: M$^\natural$-Concave Functions

Let V be a finite set. For each $S \subseteq V$, e_S denotes the characteristic vector of S defined by: $e_S(v) = 1$ if $v \in S$ and $e_S(v) = 0$ otherwise. Also define e_0 as the zero vector in $\mathbf{Z}^V$. For a vector $x \in \mathbf{Z}^V$, its positive support $\mathrm{supp}^+(x)$ and negative support $\mathrm{supp}^-(x)$ is defined by $\mathrm{supp}^+(x) = \{u \in V \mid x(u) > 0\}$ and $\mathrm{supp}^-(x) = \{u \in V \mid x(u) < 0\}$. A function $f\colon \mathbf{Z}^V \to \mathbf{R} \cup \{-\infty\}$ is called *M$^\natural$-concave* if it satisfies the following condition $\forall x, y \in \mathrm{dom}\, f$, $\forall u \in \mathrm{supp}^+(x-y), \exists v \in \mathrm{supp}^-(x-y) \cup \{0\}$:

$$f(x) + f(y) \le f(x - e_u + e_v) + f(y + e_u - e_v).$$

The above condition says that the sum of the function values at two points does not decrease as the points symmetrically move one or two steps closer to each other on the set of integral lattice points of $\mathbf{Z}^V$. This is a discrete analogue of the fact that for an ordinary concave function the sum of the function values at two points does not decrease as the points symmetrically move closer to each other on the straight line segment between the two points.

Example 1 A nonempty family $\mathcal{T}$ of subsets of V is called a *laminar family* if $X \cap Y = \emptyset$, $X \subseteq Y$ or $Y \subseteq X$ holds for every $X, Y \in \mathcal{T}$. For a laminar family $\mathcal{T}$ and a family of univariate concave functions $f_Y : \mathbf{R} \to \mathbf{R} \cup \{-\infty\}$ indexed by $Y \in \mathcal{T}$, the function $f : \mathbf{Z}^V \to \mathbf{R} \cup \{-\infty\}$ defined by

$$f(x) = \sum_{Y \in \mathcal{T}} f_Y\left(\sum_{v \in Y} x(v)\right) \qquad (\forall x \in \mathbf{Z}^V)$$

is M$^\natural$-concave. The stable marriage problem can be formulated as Problem 1 by using value functions of this type.

Example 2 For the independence family $\mathcal{I} \subseteq 2^V$ of a matroid on V and $w \in \mathbf{R}^V$, the function $f\colon \mathbf{Z}^V \to \mathbf{R} \cup \{-\infty\}$ defined by

$$f(x) = \begin{cases} \sum_{u \in X} w(u) & \text{if } x = e_X \text{ for some } X \in \mathcal{I} \\ -\infty & \text{otherwise} \end{cases} \qquad (\forall x \in \mathbf{Z}^V)$$

is M$^\natural$-concave. Fleiner [4] showed that there always exists a stable allocation for value functions of this type.

Theorem 1 ([6]) *Assume that the value functions f_k $(k \in M \cup W)$ are M$^\natural$-concave satisfying **(A)**. Then a feasible allocation x is stable if and only if there exist $z_M = (z_{(i)} \mid i \in M) \in (\mathbf{Z} \cup \{+\infty\})^E$ and $z_W = (z_{(j)} \mid j \in W) \in (\mathbf{Z} \cup \{+\infty\})^E$ such that*

$$x_{(i)} \in \arg\max\{f_i(y) \mid y \leq z_{(i)}\} \qquad (\forall i \in M)\,, \tag{7}$$

$$x_{(j)} \in \arg\max\{f_j(y) \mid y \leq z_{(j)}\} \qquad (\forall j \in W)\,, \tag{8}$$

$$z_M(e) = +\infty \text{ or } z_W(e) = +\infty \qquad (\forall e \in E)\,, \tag{9}$$

where $\arg\max\{f_i(y) \mid y \leq z_{(i)}\}$ denotes the set of all maximizers of f_i under the constraints $y \leq z_{(i)}$.

Theorem 2 ([3]) *Assume that the value functions f_k $(k \in M \cup W)$ are M$^\natural$-concave satisfying **(A)**. Then there always exists a stable allocation.*

Eguchi, Fujishige, and Tamura [3] proved Theorem 2 by showing that the following algorithm finds a feasible allocation x, and z_M, z_W satisfying (7), (8), and (9).

```
Algorithm EXTENDED-GS
Input: M$^\natural$-concave functions $f_M, f_W$ with $f_M(x) = \sum_{i \in M} f_i(x_{(i)})$ and $f_W(x) = \sum_{j \in W} f_j(x_{(j)})$ ;
Output: $(x, z_M, z_W)$ satisfying (7), (8), and (9);
    $z_M := (+\infty, \cdots, +\infty), z_W := x_W := \mathbf{0}$;
    repeat{
        let $x_M$ be any element in
        $\arg\max\{f_M(y) \mid x_W \leq y \leq z_M\}$ ;
        let $x_W$ be any element in
        $\arg\max\{f_W(y) \mid y \leq x_M\}$ ;
        for each $e \in E$ with $x_M(e) > x_W(e)$ {
            $z_M(e) := x_W(e)$ ;
            $z_W(e) := +\infty$ ;
        } ;
    } until $x_M = x_W$ ;
    return $(x_M, z_M, z_W \vee x_M)$.
```

Here $z_W \vee x_M$ is defined by $(z_W \vee x_M)(e) = \max\{z_W(e), x_M(e)\}$ for all $e \in E$.

Applications

Abraham, Irving, and Manlove [1] dealt with a student-project allocation problem which is a concrete example of models in [4] and [3], and discussed the structure of stable allocations.

Fleiner [5] generalized the stable marriage problem and its extension in [4] to a wide framework, and showed the existence of a stable allocation by using a fixed point theorem.

Fujishige and Tamura [6] proposed a common generalization of the stable marriage problem and the assignment game defined by Shapley and Shubik [10] by utilizing M$^\natural$-concave functions, and gave a constructive proof of the existence of a stable allocation.

Open Problems

Algorithm EXTENDED-GS solves the maximization problem of an M$^\natural$-concave function in each iteration. A maximization problem of an M$^\natural$-concave function f on E can be solved in polynomial time in $|E|$ and $\log L$, where $L = \max\{||x - y||_\infty \mid x, y \in \operatorname{dom} f\}$, provided that the function value $f(x)$ can be calculated in constant time for each x [11,12]. Eguchi, Fujishige, and Tamura [3] showed that EXTENDED-GS terminates after at most L iterations, where L is defined by $\{||x||_\infty \mid x \in \operatorname{dom} f_M\}$ in this case, and there exist a series of instances in which EXTENDED-GS requires numbers of iterations proportional to L. On the other hand, Baïou and Balinski [2] gave a polynomial time algorithm in $|E|$ for the special case where f_M and f_W are linear on rectangular domains. Whether a stable allocation for the general case can be found in polynomial time in $|E|$ and $\log L$ or not is open.

Cross References

- ▶ Assignment Problem
- ▶ Hospitals/Residents Problem
- ▶ Optimal Stable Marriage
- ▶ Stable Marriage
- ▶ Stable Marriage with Ties and Incomplete Lists

Recommended Reading

1. Abraham, D.J., Irving, R.W., Manlove, D.F.: Two Algorithms for the Student-Project Allocation Problem. J. Discret. Algorithms **5**, 73–90 (2007)
2. Baïou, M., Balinski, M.: Erratum: The Stable Allocation (or Ordinal Transportation) Problem. Math. Oper. Res. **27**, 662–680 (2002)
3. Eguchi, A., Fujishige, S., Tamura, A.: A generalized Gale-Shapley algorithm for a discrete-concave stable-marriage model. In:

Ibaraki, T., Katoh, N., Ono, H. (eds.) Algorithms and Computation: 14th International Symposium, ISAAC2003. LNCS, vol. 2906, pp. 495–504. Springer, Berlin (2003)
4. Fleiner, T.: A matroid generalization of the stable matching polytope. In: Gerards, B., Aardal K. (eds.) Integer Programming and Combinatorial Optimization: 8th International IPCO Conference. LNCS, vol. 2081, pp. 105–114. Springer, Berlin (2001)
5. Fleiner, T.: A Fixed Point Approach to Stable Matchings and Some Applications. Math. Oper. Res. **28**, 103–126 (2003)
6. Fujishige, S., Tamura, A.: A Two-Sided Discrete-Concave Market with Bounded Side Payments: An Approach by Discrete Convex Analysis. Math. Oper. Res. **32**, 136–155 (2007)
7. Gale, D., Shapley, S.L.: College admissions and the stability of marriage. Am. Math. Mon. **69**, 9–15 (1962)
8. Murota, K.: Discrete Convex Analysis. Math. Program. **83**, 313–371 (1998)
9. Murota, K.: Discrete Convex Analysis. Soc. Ind. Appl. Math. Philadelphia (2003)
10. Shapley, S.L., Shubik, M.: The Assignment Game I: The Core. Int. J. Game. Theor. **1**, 111–130 (1971)
11. Shioura, A.: Fast Scaling Algorithms for M-convex Function Minimization with Application to the Resource Allocation Problem. Discret. Appl. Math. **134**, 303–316 (2004)
12. Tamura, A.: Coordinatewise Domain Scaling Algorithm for M-convex Function Minimization. Math. Program. **102**, 339–354 (2005)

Stable Marriage with Ties and Incomplete Lists

2007; Iwama, Miyazaki, Yamauchi

Kazuo Iwama[1], Shuichi Miyazaki[2]
[1] School of Informatics, Kyoto University, Kyoto, Japan
[2] Academic Center for Computing and Media Studies, Kyoto University, Kyoto, Japan

Keywords and Synonyms

Stable matching problem

Problem Definition

In the original setting of the stable marriage problem introduced by Gale and Shapley [2], each preference list has to include all members of the other party, and furthermore, each preference list must be totally ordered (see entry ▶ Stable Marriage also).

One natural extension of the problem is then to allow persons to include ties in preference lists. In this extension, there are three variants of the stability definition, super-stability, strong stability, and weak stability (see below for definitions). In the first two stability definitions, there are instances that admit no stable matching, but there is a polynomial-time algorithm in each case that determines if a given instance admits a stable matching, and finds one if exists [8]. On the other hand, in the case of weak stability, there always exists a stable matching and one can be found in polynomial time.

Another possible extension is to allow persons to declare unacceptable partners, so that preference lists may be incomplete. In this case, every instance admits at least one stable matching, but a stable matching may not be a perfect matching. However, if there are two or more stable matchings for one instance, then all of them have the same size [3].

The problem treated in this entry allows both extensions simultaneously, which is denoted as SMTI (Stable Marriage with Ties and Incomplete lists).

Notations

An instance I of SMTI comprises n men, n women and each person's preference list that may be incomplete and may include ties. If a man m includes a woman w in his list, w is *acceptable* to m. $w_i \succ_m w_j$ means that m strictly prefers w_i to w_j in I. $w_i =_m w_j$ means that w_i and w_j are tied in m's list (including the case $w_i = w_j$). The statement $w_i \succeq_m w_j$ is true if and only if $w_i \succ_m w_j$ or $w_i =_m w_j$. Similar notations are used for women's preference lists. A matching M is a set of pairs (m, w) such that m is acceptable to w and vice versa, and each person appears at most once in M. If a man m is matched with a woman w in M, it is written as $M(m) = w$ and $M(w) = m$.

A man m and a woman w are said to form a *blocking pair for weak stability* for M if they are not partners in M but by matching them, both become better off, namely, (i) $M(m) \neq w$ but m and w are acceptable to each other, (ii) $w \succ_m M(m)$ or m is single in M, and (iii) $m \succ_w M(w)$ or w is single in M.

Two persons x and y are said to form a *blocking pair for strong stability* for M if they are not partners in M but by matching them, one becomes better off, and the other does not become worse off, namely, (i) $M(x) \neq y$ but x and y are acceptable to each other, (ii) $y \succ_x M(x)$ or x is single in M, and (iii) $x \succeq_y M(y)$ or y is single in M.

A man m and a woman w are said to form a *blocking pair for super-stability* for M if they are not partners in M but by matching them, neither become worse off, namely, (i) $M(m) \neq w$ but m and w are acceptable to each other, (ii) $w \succeq_m M(m)$ or m is single in M, and (iii) $m \succeq_w M(w)$ or w is single in M.

A matching M is called *weakly stable* (*strongly stable* and *super-stable*, respectively) if there is no blocking pair for weak (strong and super, respectively) stability for M.

Problem 1 (SMTI)
INPUT: *n men, n women, and each person's preference list.*
OUTPUT: *A stable matching.*

Problem 2 (MAX SMTI)
INPUT: *n men, n women, and each person's preference list.*
OUTPUT: *A stable matching of maximum size.*

The following problem is a restriction of MAX SMTI in terms of the length of preference lists:

Problem 3 ((p, q)-MAX SMTI)
INPUT: *n men, n women, and each person's preference list, where each man's preference list includes at most p women, and each woman's preference list includes at most q men.*
OUTPUT: *A stable matching of maximum size.*

Definition of Approximation Ratios

A goodness measure of an approximation algorithm T for a maximization problem is defined as follows: the *approximation ratio* of T is $\max\{opt(x)/T(x)\}$ over all instances x of size N, where $opt(x)$ and $T(x)$ are the size of the optimal and the algorithm's solution, respectively.

Key Results

SMTI and MAX SMTI in Super-Stability and Strong Stability

Theorem 1 ([16]) *There is an $O(n^2)$-time algorithm that determines if a given SMTI instance admits a super-stable matching, and finds one if exists.*

Theorem 2 ([15]) *There is an $O(n^3)$-time algorithm that determines if a given SMTI instance admits a strongly stable matching, and finds one if exists.*

It is shown that all stable matchings for a fixed instance are of the same size [16]. So, the above theorems imply that MAX SMTI can also be solved in the same time complexity.

SMTI and MAX SMTI in Weak Stability

In the case of weak stability, every instance admits at least one stable matching, but one instance can have stable matchings of different sizes. If the size is not important, a stable matching can be found in polynomial time by breaking ties arbitrarily and applying the Gale-Shapley algorithm.

Theorem 3 *There is an $O(n^2)$-time algorithm that finds a weakly stable matching for a given SMTI instance.*

However, if larger stable matchings are required, the problem becomes hard.

Theorem 4 ([5,7,12,17]) *MAX SMTI is NP-hard, and cannot be approximated within $21/19 - \epsilon$ for any positive constant ϵ, unless $P = NP$. ($21/19 \simeq 1.105$)*

The current best approximation algorithm is a local search type algorithm.

Theorem 5 ([13]) *There is a polynomial-time approximation algorithm for MAX SMTI, whose approximation ratio is at most $15/8 (= 1.875)$.*

There are a couple of approximation algorithms for restricted inputs.

Theorem 6 ([6]) *There is a polynomial-time randomized approximation algorithm for MAX SMTI whose expected approximation ratio is at most $10/7 (\simeq 1.429)$, if in a given instance, ties appear in only one side and the length of each tie is two.*

Theorem 7 ([6]) *There is a polynomial-time randomized approximation algorithm for MAX SMTI whose expected approximation ratio is at most $7/4 (= 1.75)$, if in a given instance, the length of each tie is two.*

Theorem 8 ([7]) *There is a polynomial-time approximation algorithm for MAX SMTI whose approximation ratio is at most $2/(1 + L^{-2})$, if in a given instance, ties appear in only one side and the length of each tie is at most L.*

Theorem 9 ([7]) *There is a polynomial-time approximation algorithm for MAX SMTI whose approximation ratio is at most $13/7 (\simeq 1.858)$, if in a given instance, the length of each tie is two.*

(p, q)-MAX SMTI in Weak Stability

Irving et al. show the boundary between P and NP in terms of the length of preference lists.

Theorem 10 ([11]) *$(2, \infty)$-MAX SMTI is solvable in time $O(n^{\frac{3}{2}} \log n)$.*

Theorem 11 ([11]) *(3,4)-MAX SMTI is NP-hard, and cannot be approximated within some constant $\delta (> 1)$, unless $P = NP$.*

Recently, Manlove proved NP-hardness of (3,3)-MAX SMTI [18].

Applications

One of the most famous applications of the stable marriage problem is a centralized assignment system between medical students (residents) and hospitals. This is an extension of the stable marriage problem to a many-one variant: Each hospital declares the number of residents it can accept, which may be more than one, while each resident has to be assigned to at most one hospital. Actually, there are several applications in the world, known as NRMP in the US [4], CaRMS in Canada [1], SPA in Scotland [9,10], and JRMP in Japan [14]. One of the optimization criteria is clearly the number of matched residents. In a real-world application such as the above residents matching, hospitals and residents tend to submit short preference lists that include ties, in which case, the problem can be naturally considered as MAX SMTI.

Open Problems

One apparent open problem is to narrow the gap of approximability of MAX SMTI in weak stability, namely, between $15/8(= 1.875)$ and $21/19(\simeq 1.105)$ for general case. The same problem can be considered for restricted instances. The reduction shown in [7] creates instances where ties appear in only one side, and the length of ties is two. So, considering Theorem 8 for $L = 2$, there is a gap between $8/5(= 1.6)$ and $21/19(\simeq 1.105)$ in this case. It is shown in [7] that if the $2 - \epsilon$ lower bound (for any positive constant ϵ) on the approximability of Minimum Vertex Cover is derived, the same reduction shows the $5/4 - \delta$ lower bound (for any positive constant δ) on the approximability of MAX SMTI.

Cross References

- ▶ Assignment Problem
- ▶ Hospitals/Residents Problem
- ▶ Optimal Stable Marriage
- ▶ Ranked Matching
- ▶ Stable Marriage
- ▶ Stable Marriage and Discrete Convex Analysis
- ▶ Stable Partition Problem

Recommended Reading

1. Canadian Resident Matching Service (CaRMS) http://www.carms.ca/. Accessed 27 Feb 2008, JST
2. Gale, D., Shapley, L.S.: College admissions and the stability of marriage. Am. Math. Monthly **69**, 9–15 (1962)
3. Gale, D., Sotomayor, M.: Some remarks on the stable matching problem. Discret. Appl. Math. **11**, 223–232 (1985)
4. Gusfield, D., Irving, R.W.: The Stable Marriage Problem: Structure and Algorithms. MIT Press, Boston, MA (1989)
5. Halldórsson, M.M., Irving, R.W., Iwama, K., Manlove, D.F., Miyazaki, S., Morita, Y., Scott, S.: Approximability results for stable marriage problems with ties. Theor. Comput. Sci. **306**, 431–447 (2003)
6. Halldórsson, M.M., Iwama, K., Miyazaki, S., Yanagisawa, H.: Randomized approximation of the stable marriage problem. Theor. Comput. Sci. **325**(3), 439–465 (2004)
7. Halldórsson, M.M., Iwama, K., Miyazaki, S., Yanagisawa, H.: Improved approximation of the stable marriage problem. Proc. ESA 2003. LNCS 2832, pp. 266–277. (2003)
8. Irving, R.W.: Stable marriage and indifference. Discret. Appl. Math. **48**, 261–272 (1994)
9. Irving, R.W.: Matching medical students to pairs of hospitals: a new variation on a well-known theme. Proc. ESA 98. LNCS 1461, pp. 381–392. (1998)
10. Irving, R.W., Manlove, D.F., Scott, S.: The hospitals/residents problem with ties. Proc. SWAT 2000. LNCS 1851, pp. 259–271. (2000)
11. Irving, R.W., Manlove, D.F., O'Malley, G.: Stable marriage with ties and bounded length preference lists. Proc. the 2nd Algorithms and Complexity in Durham workshop, Texts in Algorithmics, College Publications (2006)
12. Iwama, K., Manlove, D.F., Miyazaki, S., Morita, Y.: Stable marriage with incomplete lists and ties. Proc. ICALP 99. LNCS 1644, pp. 443–452. (1999)
13. Iwama, K., Miyazaki, S., Yamauchi, N.: A 1.875-approximation algorithm for the stable marriage problem. Proc, SODA 2007, pp. 288–297. (2007)
14. Japanese Resident Matching Program (JRMP) http://www.jrmp.jp/
15. Kavitha, T., Mehlhorn, K., Michail, D., Paluch, K.: Strongly stable matchings in time $O(nm)$ and extension to the hospitals-residents problem. Proc. STACS 2004. LNCS (2996), pp. 222–233. (2004)
16. Manlove, D.F.: Stable marriage with ties and unacceptable partners. Technical Report no. TR-1999-29 of the Computing Science Department of Glasgow University (1999)
17. Manlove, D.F., Irving, R.W., Iwama, K., Miyazaki, S., Morita, Y.: Hard variants of stable marriage. Theor. Comput. Sci. **276**(1–2), 261–279 (2002)
18. Manlove, D.F.: private communication (2006)

Stable Matching

- ▶ Market Games and Content Distribution
- ▶ Stable Marriage
- ▶ Stable Marriage and Discrete Convex Analysis
- ▶ Stable Marriage with Ties and Incomplete Lists

Stable Partition Problem

2002; Cechlárová, Hajduková

KATARÍNA CECHLÁROVÁ
Faculty of Science, Institute of Mathematics,
P.J. Šafárik University, Košice, Slovakia

Keywords and Synonyms

In the economists community these models are often referred to as *Coalition formation games* [4,7], or *Hedonic* *games* [3,6,16]; some variants correspond to the *Directed cycle cover* problems [1]. Important special cases are the *Stable Matching Problems* [17]. .

Problem Definition

In the Stable Partition Problem a set of participants has to be split into several disjoint sets called *coalitions*. The resulting partition should fulfill some stability requirements that take into account the preferences of participants.

Various variants of this problem arise if the participants are required to express their preferences over all the possible coalitions to which they could belong or when only preferences over other players are given and those are then extended to preferences over coalitions. Sometimes one seeks rather a permutation of players and the partition is given by the cycles of the permutation [1, 19].

Notation

An instance of the Stable Partition Problem (SPP for short) is a pair $(N,\mathcal{P})$, where N is a finite set of participants and $\mathcal{P}$ the collection of their preferences, called the *preference profile*. If the preferences of participants are given as linearly ordered lists of the coalitions to which a particular participant can belong (i. e. participant i writes a list of subsets of N that contain i), we say that the instance of the SPP is in the *LC form* (list of coalitions). A special case of the SPP in the LC form is obtained when participants do not care about the actual content of the coalitions, only about their sizes. Preferences are then called *anonymous*.

A more succinct representation is obtained when each participant i linearly orders only individual participants, or more precisely, a subset of them – these are *acceptable* for i. In this case the SPP is in the *LP form* (list of participants). With the exception of Stable Matchings, when the obtained partitions are allowed to contain only singletons or a two-element sets, preferences over participants have to be extended to preferences over coalitions. Algorithmically, the most intensively studied are the following extensions:

- $\mathcal{B}$**-preferences –** a participant orders coalitions first on the basis of the most preferred (briefly best) member of the coalition, and if those are equal or tied, the coalition with smaller cardinality is preferred;
- $\mathcal{W}$**-preferences –** a participant orders coalitions on the basis of the least preferred (briefly worst) member of the coalition;
- $\mathcal{BW}$**-preferences –** a participant orders coalitions first on the basis of the best member of the coalition, and if those are equal or tied, the coalition with a more preferred worst member is preferred.

The above preferences are said to be *strict*, if the original preferences over individuals are strict linear orders and they are called *dichotomous* if all acceptable participants are tied in each preference list. The presence of ties very often leads to different computational results compared to the case with strict preferences.

In *additively separable* preferences it is supposed that for each $i \in N$ there exists a function $v_i : N \to \mathbb{R}$ such that i prefers a coalition S to coalition T if and only if $\sum_{j\in S} v_i(j) > \sum_{j\in T} v_i(j)$. Additively separable preferences and their various variants are studied in [7].

Another approach is presented in [16]. The authors call these preferences *simple* and it is supposed that for each participant i a set F_i of friends and a set E_i of enemies are given. A participant i has *appreciation of friends* when he prefers a coalition S to a coalition T if $|S \cap F_i| > |T \cap F_i|$ and he has *aversion against enemies* when he prefers a coalition S to a coalition T if $|S \cap E_i| < |T \cap E_i|$.

Stability Definitions

Let $M(i)$ denote the set of partition M that contains participant i.

Definition 1 A set $Z \subseteq N$ is called *blocking* for partition M, if each participant $i \in Z$ prefers Z to $M(i)$. A set $Z \subseteq N$ is called *weakly blocking* for partition M, if each participant $i \in Z$ prefers Z to $M(i)$ or is indifferent between Z and $M(i)$ and at least one participant $j \in Z$ prefers Z to $M(j)$.

A participant i is said to be *covered* if $|M(i)| \geq 2$.

In the literature, several different stability definitions were studied, including *Nash stability, individual stability, contractual individual stability, Pareto optimality* etc. An interested reader can consult [4] or [6]. Algorithmically, the most deeply studied notions are the core and the strong core.

Definition 2 A partition M is called a *core* partition, if there is no blocking set for M. A partition M is called a *strong core* partition, if there is no weakly blocking set for M.

Problems

Several decision or computational problems arise in the context of the SPP:

- StabilityTest: Given $(N,\mathcal{P})$ and a partition M of N, is M stable?

- EXISTENCE: Does a stable partition for a given $(N,\mathcal{P})$ exist?
- CONSTRUCTION: If a stable partition for a given $(N,\mathcal{P})$ exists, find one.
- STRUCTURE: Describe the structure of stable partitions for a given $(N,\mathcal{P})$.

Their complexity depends on the particular type of preferences used.

Key Results

SPP in LC Form

EXISTENCE for core partitions is NP-complete even when the given preferences over coalitions are strict or anonymous [3].

$\mathcal{W}$-preferences

The SPP with strict $\mathcal{W}$-preferences has many features similar to the Stable Roommates Problem [17]. First, each core partition set contains at most two participants and if a blocking set exists, then there is a blocking set of size at most 2, hence STABILITYTEST is polynomial. EXISTENCE and CONSTRUCTION are polynomial in the strict preferences case [11], which can be shown using an extension of Irving's Stable Roommates Algorithm (discussed in detail in [17]). This algorithm can also be used to derive some results for STRUCTURE. In the case of ties, EXISTENCE is NP-complete and a complete solution to STRUCTURE is not available [11].

$\mathcal{B}$-preferences

A polynomial algorithm for STABILITYTEST is given in [9]. For strict $\mathcal{B}$-preferences a core as well as strong core partition always exists and one can be found by the Top Trading Cycles algorithm attributed to Gale in [19] (an implementation of this algorithm of time complexity $O(m)$, where m is the total length of the preference lists of all participants, was described in [2]). However, if preferences of participants contain ties, EXISTENCE is NP-complete for both core and strict core [10]. In the dichotomous case, a core partition can be constructed in polynomial time, but EXISTENCE for strong core is NP-complete [8].

Very little is known about the STRUCTURE. Several questions about the existence of core partitions with special properties are shown to be NP-hard even for the strict preferences case [15]:

- Does a core partition M exist, such that $|M(i)| < |T(i)|$ for each participant i, where T is the partition obtained by the Top Trading Cycles algorithm?
- Does a core partition M exist, such that $|M(i)| \leq 3$ for each participant i?
- Does a core partition M exist that covers all participants?

Moreover, the maximum number of participants covered by a core partition is not approximable within $n^{1-\varepsilon}$ [5].

$\mathcal{BW}$-preferences

In the strict preferences case a core partition always exists and one can be obtained by the Top Trading Cycles algorithm. However, if preferences contain ties, EXISTENCE is NP-hard [12]. STABILITYTEST remains open.

Simple Preferences

If all the participants have aversion to enemies, a core partition always exists, but CONSTRUCTION is NP-hard. In the appreciation-of-friends case, a strong core partition always exists and CONSTRUCTION can be solved in $O(n^3)$ time, where n is the number of participants [16].

Applications

Stable partitions give rise to various economic and game theoretical models. They appear in the study of exchange economies with discrete commodities [19], in barter exchange markets [20], or in the study of formation of countries [14]. A recent application concerns exchange of kidneys for transplantation between willing donors and their incompatible intended recipients [18]. In this context, the use of $\mathcal{B}$-preferences was suggested in [8], as they express the wish of each patient for the best suitable kidney as well as his desire for the shortest possible exchange cycle.

Open Problems

Because of the great number of variants, a lot of open problems exist. In almost all cases, STRUCTURE is not satisfactorily solved. For instances with no stable partition, one may seek one that minimizes the number of participants who have an incentive to deviate. Parallel algorithms were also not studied.

Experimental Results

In the context of kidney exchange, Roth et al. in [18] performed extensive experiments with the Top Trading Cycles algorithm on simulated patients' data. The number of covered participants and sizes of the obtained partition sets were recorded. The structure of core partitions for $\mathcal{B}$-preferences was studied in [15]. Two heuristics were tested. The starting point was the stable partition obtained by the Top Trading Cycles algorithm. Heuristic Cut-Cycle tried to split at least one of the obtained partition sets, Cut-and-Add tried to add an uncovered participant to an existing partition set on condition that the new partition remained in the core. It was shown that as the total number of participants grows, the percentage of participants uncovered in the Top Trading Cycles partition decreases and the percentage of successes of both heuristics grows.

Cross References

► Hospitals/Residents Problem
► Optimal Stable Marriage
► Ranked Matching
► Stable Marriage
► Stable Marriage with Ties and Incomplete Lists

Recommended Reading

1. Abraham, D., Blum, A., Sandholm, T.: Clearing algorithms for barter exchange markets: Enabling nationwide kidney exchanges. EC'07, June 11–15, 2007, San Diego, California
2. Abraham, D., Cechlárová, K., Manlove, D., Mehlhorn, K.: Pareto-optimality in house allocation problems. In: Fleischer, R., Trippen, G. (eds.) Lecture Notes in Comp. Sci. Vol. 3341/2004, Algorithms and Computation, 14th Int. Symposium ISAAC 2004, pp. 3–15. Hong Kong, December 2004
3. Ballester, C.: NP-completeness in Hedonic Games. Games. Econ. Behav. **49**(1), 1–30 (2004)
4. Banerjee, S., Konishi, H., Sönmez, T.: Core in a simple coalition formation game. Soc. Choice. Welf. **18**, 135–153 (2001)
5. Biró, P., Cechlárová, K.: Inapproximability of the kidney exchange problem. Inf. Proc. Lett. **101**(5), 199–202 (2007)
6. Bogomolnaia, A., Jackson, M.O.: The Stability of Hedonic Coalition Structures. Games. Econ. Behav. **38**(2), 201–230 (2002)
7. Burani, N., Zwicker, W.S.: Coalition formation games with separable preferences. Math. Soc. Sci. **45**, 27–52 (2003)
8. Cechlárová, K., Fleiner, T., Manlove, D.: The kidney exchange game. In: Zadnik-Stirn, L., Drobne, S. (eds.) Proc. SOR '05, pp. 77–83. Nova Gorica, September 2005
9. Cechlárová, K., Hajduková, J.: Stability testing in coalition formation games. In: Rupnik, V., Zadnik-Stirn, L., Drobne, S. (eds.) Proceedings of SOR'99, pp. 111–116. Predvor, Slovenia (1999)
10. Cechlárová, K., Hajduková, J.: Computational complexity of stable partitions with $\mathcal{B}$-preferences. Int. J. Game. Theory **31**(3), 353–364 (2002)
11. Cechlárová, K., Hajduková, J.: Stable partitions with $\mathcal{W}$-preferences. Discret. Appl. Math. **138**(3), 333–347 (2004)
12. Cechlárová, K., Hajduková, J.: Stability of partitions under WB-preferences and BW-preferences. Int. J. Inform. Techn. Decis. Mak. Special Issue on Computational Finance and Economics. **3**(4), 605–614 (2004)
13. Cechlárová, K., Romero-Medina, A.: Stability in coalition formation games. Int. J. Game. Theor. **29**, 487–494 (2001)
14. Cechlárová, K., Dahm, M., Lacko, V.: Efficiency and stability in a discrete model of country formation. J. Glob. Opt. **20**(3–4), 239–256 (2001)
15. Cechlárová, K., Lacko, V.: The Kidney Exchange problem: How hard is it to find a donor? IM Preprint A4/2006, Institute of Mathematics, P.J. Šafárik University, Košice, Slovakia, (2006)
16. Dimitrov, D., Borm, P., Hendrickx, R., Sung, S. Ch.: Simple priorities and core stability in hedonic games. Soc. Choice. Welf. **26**(2), 421–433 (2006)
17. Gusfield, D., Irving, R.W.: The Stable Marriage Problem. Structure and Algorithms. MIT Press, Cambridge (1989)
18. Roth, A., Sönmez, T., Ünver, U.: Kidney Exchange. Quarter. J. Econ. **119**, 457–488 (2004)
19. Shapley, L., Scarf, H.: On cores and indivisibility. J. Math. Econ. **1**, 23–37 (1974)
20. Yuan, Y.: Residence exchange wanted: A stable residence exchange problem. Eur. J. Oper. Res. **90**, 536–546 (1996)

Stackelberg Games: The Price of Optimum

2006; Kaporis, Spirakis

Alexis Kaporis[1], Paul Spirakis[2]
[1] Department of Computer Engineering & Informatics, University of Patras, Patras, Greece
[2] Computer Engineering and Informatics, Research and Academic Computer Technology Institute, Patras University, Patras, Greece

Keywords and Synonyms

Cournot game; Coordination ratio

Problem Definition

Stackelberg games [15] may model the interplay amongst an authority and rational individuals that selfishly demand resources on a large scale network. In such a game, the authority *(Leader)* of the network is modeled by a distinguished player. The selfish users *(Followers)* are modeled by the remaining players.

It is well known that selfish behavior may yield a *Nash Equilibrium* with cost arbitrarily higher than the optimum one, yielding unbounded *Coordination Ratio* or *Price of Anarchy (PoA)* [7,13]. Leader plays his strategy first assigning a portion of the total demand to some resources of the network. Followers observe and react selfishly assigning their demand to the most appealing resources. Leader aims to drive the system to an a posteriori Nash equilibrium with cost close to the overall optimum one [4,6,8,10]. Leader may also eager for his own rather than system's performance [2,3].

A Stackelberg game can be seen as a special, and easy [6] to implement, case of *Mechanism Design*. It avoids the complexities of either computing taxes or assigning prices, or even designing the network at hand [9]. However, a central authority capable to control the overall demand on the resources of a network may be unrealistic in networks which evolute and operate under the effect of many and diversing economic entities. A realistic way [4] to act centrally even in large nets could be via *Virtual Private Networks (VPNs)* [1]. Another flexible way is to combine such strategies with *Tolls* [5,14].

A dictator controlling the entire demand optimally on the resources surely yields $PoA = 1$. On the other hand, rational users do prefer a liberal world to live. Thus, it is important to compute the optimal Leader-strategy which controls the *minimum* of the resources *(Price of Optimum)* and yields $PoA = 1$. What is the complexity of computing the Price of Optimum? This is not trivial to answer, since the Price of Optimum depends crucially on computing an optimal Leader strategy. In particular, [6] proved that computing the optimal Leader strategy is hard.

The central result of this lemma is Theorem 5. It says that on nonatomic flows and arbitrary *s*-*t* networks & latencies, computing the minimum portion of flow and Leader's optimal strategy sufficient to induce $PoA = 1$ is easy [10].

Problem $(G(V, E), s, t \in V, r)$
INPUT: *Graph G,* $\forall e \in E$ *latency* ℓ_e*, flow r, a source-destination pair (s, t) of vertices in V.*
OUTPUT: *(i) The minimum portion* α_G *of the total flow r sufficient for an optimal Stackelberg strategy to induce the optimum on G. (ii) The optimal Stackelberg strategy.*

Models & Notations

Consider a graph $G(V, E)$ with parallel edges allowed. A number of rational and selfish users wish to route from a given source s to a destination node t an amount of flow r. Alternatively, consider a partition of users in k commodities, where user(s) in commodity i wish to route flow r_i through a source-destination pair (s_i, t_i), for each $i = 1, \ldots, k$. Each edge $e \in E$ is associated to a latency function $\ell_e()$, positive, differentiable and strictly increasing on the flow traversing it.

Nonatomic Flows There are infinitely many users, each routing his infinitesimally small amount of the total flow r_i from a given source s_i to a destination vertex t_i in graph $G(V, E)$. A flow f is an assignment of jobs f_e on each edge $e \in E$. The cost of the injected flow f_e (satisfying the standard constraints of the corresponding network-flow problem) that traverses edge $e \in E$ equals $c_e(f_e) = f_e \times \ell_e(f_e)$. It is assumed that on each edge e the cost is convex with respect the injected flow f_e. The overall system's cost is the sum $\sum_{e \in E} f_e \times \ell_e(f_e)$ of all edge-costs in G. Let $f_{\mathcal{P}}$ the amount of flow traversing the s_i-t_i path $\mathcal{P}$. The latency $\ell_{\mathcal{P}}(f)$ of s_i-t_i path $\mathcal{P}$ is the sum $\sum_{e \in \mathcal{P}} \ell_e(f_e)$ of latencies per edge $e \in \mathcal{P}$. The cost $C_{\mathcal{P}}(f)$ of s_i-t_i path $\mathcal{P}$ equals the flow $f_{\mathcal{P}}$ traversing it multiplied by path-latency $\ell_{\mathcal{P}}(f)$. That is, $C_{\mathcal{P}}(f) = f_{\mathcal{P}} \times \sum_{e \in \mathcal{P}} \ell_e(f_e)$.

In an Nash equilibrium, all s_i-t_i paths traversed by nonatomic users in part i have a common latency, which is at most the latency of any untraversed s_i-t_i path. More formally, for any part i and any pair $\mathcal{P}_1, \mathcal{P}_2$ of s_i-t_i paths, if $f_{\mathcal{P}_1} > 0$ then $\ell_{\mathcal{P}_1}(f) \leq \ell_{\mathcal{P}_2}(f)$. By the convexity of edge-costs the Nash equilibrium is unique and computable in polynomial time given a floating-point precision. Also computable is the unique *Optimum* assignment O of flow, assigning flow o_e on each $e \in E$ and minimizing the overall cost $\sum_{e \in E} o_e \ell_e(o_e)$. However, not all optimally traversed s_i-t_i paths experience the same latency. In particular, users traversing paths with high latency have incentive to reroute towards more speedy paths. Therefore the optimal assignment is unstable on selfish behavior.

A Leader dictates a *weak* Stackelberg strategy if on each commodity $i = 1, \ldots, k$ controls a fixed α portion of flow r_i, $\alpha \in [0, 1]$. A *strong* Stackelberg strategy is more flexible, since Leader may control $\alpha_i r_i$ flow in commodity i such that $\sum_{i=1}^{k} \alpha_i = \alpha$. Let a Leader dictating flow s_e on edge $e \in E$. The a posteriori latency $\widetilde{\ell_e}(n_e)$ of edge e, with respect to the induced flow n_e by the selfish users, equals $\widetilde{\ell_e}(n_e) = \ell_e(n_e + s_e)$. In the a posteriori Nash equilibrium, all s_i-t_i paths traversed by the free selfish users in commodity i have a common latency, which is at most the latency of any selfishly untraversed path, and its cost is $\sum_{e \in E}(n_e + s_e) \times \widetilde{\ell_e}(n_e)$.

Atomic Splittable Flows There is a finite number of atomic users $1, \ldots, k$. Each user i is responsible for routing a non-negligible flow-amount r_i from a given source s_i to a destination vertex t_i in graph G. In turn, each flow-amount r_i consists of infinitesimally small jobs.

Let flow f assigning jobs f_e on each edge $e \in E$. Each edge-flow f_e is the sum of partial flows $f_e^1, \ldots, f_e^k$ injected by the corresponding users $1, \ldots, k$. That is, $f_e = f_e^1 + \cdots + f_e^k$. As in the model above, the latency on a given s_i-t_i path $\mathcal{P}$ is the sum $\sum_{e \in \mathcal{P}} \ell_e(f_e)$ of latencies per edge $e \in \mathcal{P}$. Let $f_{\mathcal{P}}^i$ be the flow that user i ships through an s_i-t_i path $\mathcal{P}$. The cost of user i on a given s_i-t_i path $\mathcal{P}$ is analogous to her path-flow $f_{\mathcal{P}}^i$ routed via $\mathcal{P}$ times the total path-latency $\sum_{e \in \mathcal{P}} \ell_e(f_e)$. That is, the path-cost equals $f_{\mathcal{P}}^i \times \sum_{e \in \mathcal{P}} \ell_e(f_e)$. The overall cost $C_i(f)$ of user i is the sum of the corresponding path-costs of all s_i-t_i paths.

In a Nash equilibrium no user i can improve his cost $C_i(f)$ by rerouting, given that any user $j \neq i$ keeps his routing fixed. Since each atomic user minimizes its cost, if the game consists of only one user then the cost of the Nash equilibrium coincides to the optimal one.

In a Stackelberg game, a distinguished atomic Leader-player controls flow r_0 and plays first assigning flow s_e on edge $e \in E$. The a posteriori latency $\widetilde{\ell}_e(x)$ of edge e on induced flow x equals $\widetilde{\ell}_e(x) = \ell_e(x + s_e)$. Intuitively, after Leader's move, the induced selfish play of the k atomic users is equivalent to atomic splittable flows on a graph where each initial edge-latency ℓ_e has been mapped to $\widetilde{\ell}_e$. In game-parlance, each atomic user $i \in \{1, \ldots, k\}$, having *fixed* Leader's strategy, computes his *best reply* against all others atomic users $\{1, \ldots, k\} \setminus \{i\}$. If n_e is the induced Nash flow on edge e this yields total cost $\sum_{e \in E}(n_e + s_e) \times \widetilde{\ell}_e(n_e)$.

Atomic Unsplittable Flows The users are finite $1, \ldots, k$ and user i is allowed to sent his non-negligible job r_i only on a *single* path. Despite this restriction, all definitions given in atomic splittable model remain the same.

Key Results

Let us see first the case of atomic splittable flows, on parallel M/M/1 links with different speeds connecting a given source-destination pair of vertices.

Theorem 1 (Korilis, Lazar, Orda [6]) *The Leader can enforce in polynomial time the network optimum if she controls flow r_0 exceeding a critical value $\underline{r}^0$.*

In the sequel, we focus on nonatomic flows on *s*-*t* graphs with parallel links. In [6] primarily were studied cases that Leader's flow cannot induce network's optimum and was shown that an optimal Stackelberg strategy is easy to compute. In this vain, if *s*-*t* parallel-links instances are restricted to ones with linear latencies of equal slope then an optimal strategy is easy [4].

Theorem 2 (Kaporis, Spirakis [4]) *The optimal Leader strategy can be computed in polynomial time on any instance (G, r, α) where G is an s-t graph with parallel-links and linear latencies of equal slope.*

Another positive result is that the optimal strategy can be approximated within $(1 + \epsilon)$ in polynomial time, given that link-latencies are polynomials with non-negative coefficients.

Theorem 3 (Kumar, Marathe [8]) *There is a fully* polynomial approximate *Stackelberg scheme that runs in $poly(m, \frac{1}{\epsilon})$ time and outputs a strategy with cost $(1 + \epsilon)$ within the optimum strategy.*

For parallel link *s*-*t* graphs with arbitrary latencies more can be achieved: in polynomial time a "threshold" value α_G is computed, sufficient for the Leader's portion to induce the optimum. The complexity of computing optimal strategies changes in a dramatic way around the critical value α_G from "hard" to "easy" (G, r, α) Stackelberg scheduling instances. Call α_G as the *Price of Optimum* for graph G.

Theorem 4 (Kaporis, Spirakis [4]) *On input an s-t parallel link graph G with arbitrary strictly increasing latencies the minimum portion α_G sufficient for a Leader to induce the optimum, as well as her optimal strategy, can be computed in polynomial time.*

As a conclusion, the Price of Optimum α_G essentially captures the hardness of instances (G, r, α). Since, for Stackelberg scheduling instances $(G, r, \alpha \geq \alpha_G)$ the optimal Leader strategy yields $PoA = 1$ and it is computed as hard as in P, while for $(G, r, \alpha < \alpha_G)$ the optimal strategy yields $PoA < 1$ and it is as easy as NP [10].

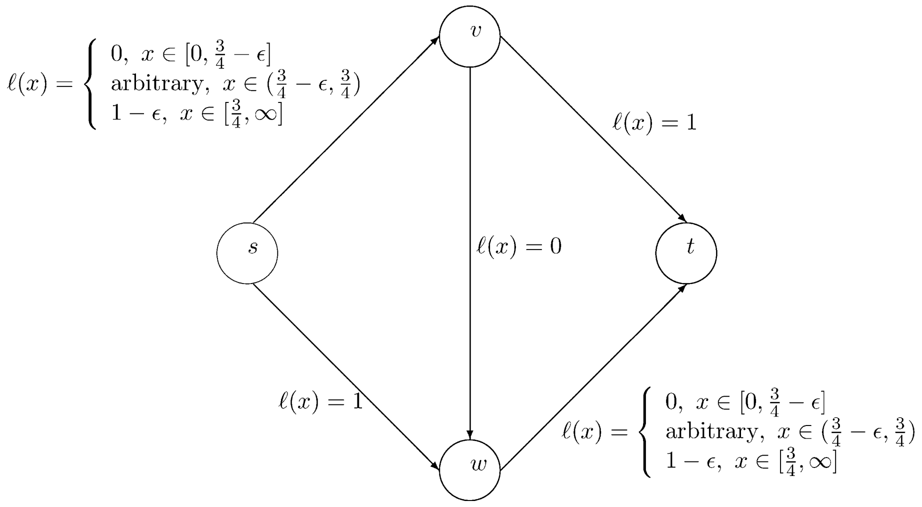
The results above are limited to parallel-links connecting a given *s*-*t* pair of vertices. Is it possible to efficiently compute the Price of Optimum for nonatomic flows on arbitrary graphs? This is not trivial to settle. Not only because it relies on computing an optimal Stackelberg strategy, which is hard to tackle [10], but also because Proposition B.3.1 in [11] ruled out previously known performance guarantees for Stackelberg strategies on general nets.

The central result of this lemma is presented below and completely resolves this question (extending Theorem 4).

Theorem 5 (Kaporis, Spirakis [4]) *On arbitrary s-t graphs G with arbitrary latencies the minimum portion α_G sufficient for a Leader to induce the optimum, as well as her optimal strategy, can be computed in polynomial time.*

Example

Consider the optimum assignment O of flow r that wishes to travel from source vertex s to sink t. O assigns flow o_e incurring latency $\ell_e(o_e)$ per edge $e \in G$. Let $\mathcal{P}_{s \to t}$ the set of all *s*-*t* paths. The *shortest paths* in $\mathcal{P}_{s \to t}$ with respect to costs $\ell_e(o_e)$ per edge $e \in G$ can be computed in polynomial time. That is, the paths that given flow assignment O attain latency: $\min_{P \in \mathcal{P}_{s \to t}} \left(\sum_{e \in P} \ell_e(o_e)\right)$ i. e., minimize their latency. It is crucial to observe that, if we want the *induced* Nash assignment by the Stackelberg strategy to attain the optimum cost, then these shortest paths are *the only choice* for selfish users that eager to travel from s to t. Furthermore, the uniqueness of the optimum assignment O determines the minimum part of flow which can be selfishly scheduled on these shortest paths. Observe that any


$\ell(x) = \begin{cases} 0, & x \in [0, \frac{3}{4} - \epsilon] \\ \text{arbitrary}, & x \in (\frac{3}{4} - \epsilon, \frac{3}{4}) \\ 1 - \epsilon, & x \in [\frac{3}{4}, \infty] \end{cases}$
v
$\ell(x) = 1$
s
$\ell(x) = 0$
t
$\ell(x) = 1$
w
$\ell(x) = \begin{cases} 0, & x \in [0, \frac{3}{4} - \epsilon] \\ \text{arbitrary}, & x \in (\frac{3}{4} - \epsilon, \frac{3}{4}) \\ 1 - \epsilon, & x \in [\frac{3}{4}, \infty] \end{cases}$

Stackelberg Games: The Price of Optimum, Figure 1
A bad example for Stackelberg routing

flow assigned by O on a non-shortest s-t path has incentive to opt for a shortest one. Then a Stackelberg strategy *must* frozen the flow on all non-shortest s-t paths.

In particular, the idea sketched above achieves coordination ratio 1 on the graph in Fig. 1. On this graph Roughgarden proved that $\frac{1}{\alpha}\times$ (optimum cost) guarantee is *not* possible for general (s, t)-networks, Appendix B.3 in [11]. The optimal edge-flows are $(r = 1)$:

$$o_{s\to v} = \frac{3}{4} - \epsilon\,,\, o_{s\to w} = \frac{1}{4} + \epsilon\,,\, o_{v\to w} = \frac{1}{2} - 2\epsilon\,,$$
$$o_{v\to t} = \frac{1}{4} + \epsilon\,,\, o_{w\to t} = \frac{3}{4} - \epsilon$$

The shortest path $P_0 \in \mathcal{P}$ with respect to the optimum O is $P_0 = s \to v \to w \to t$ (see [11] pp. 143, 5th-3th lines before the end) and its flow is $f_{P_0} = \frac{1}{2} - 2\epsilon$. The non shortest paths are: $P_1 = s \to v \to t$ and $P_2 = s \to w \to t$ with corresponding optimal flows: $f_{P_1} = \frac{1}{4} + \epsilon$ and $f_{P_2} = \frac{1}{4} + \epsilon$. Thus the Price of Optimum is

$$f_{P_1} + f_{P_2} = \frac{1}{2} + 2\epsilon = r - f_{P_0}$$

Applications

Stackelberg strategies are widely applicable in networking [6], see also Section 6.7 in [12].

Open Problems

It is important to extend the above results on atomic unsplittable flows.

Cross References

- ▶ Algorithmic Mechanism Design
- ▶ Best Response Algorithms for Selfish Routing
- ▶ Facility Location
- ▶ Non-approximability of Bimatrix Nash Equilibria
- ▶ Price of Anarchy
- ▶ Selfish Unsplittable Flows: Algorithms for Pure Equilibria

Recommended Reading

1. Birman, K.: Building Secure and Reliable Network Applications. Manning, (1996)
2. Douligeris, C., Mazumdar, R.: Multilevel flow control of Queues. In: Johns Hopkins Conference on Information Sciences, Baltimore, 22–24 Mar 1989 (2006)
3. Economides, A., Silvester, J.: Priority load sharing: an approach using stackelberg games. In: 28th Annual Allerton Conference on Communications, Control and Computing (1990)
4. Kaporis, A., Spirakis, P.G.: Stackelberg games on arbitrary networks and latency functions. In: 18th ACM Symposium on Parallelism in Algorithms and Architectures (2006)
5. Karakostas, G., Kolliopoulos, G.: Stackelberg strategies for selfish routing in general multicommodity networks. Technical report, Advanced Optimization Laboratory, McMaster Univercity (2006) AdvOL2006/08, 2006-06-27
6. Korilis, Y.A., Lazar, A.A., Orda, A.: Achieving network optima using stackelberg routing strategies. IEEE/ACM Trans. Netw. **5**(1), 161–173 (1997)
7. Koutsoupias, E., Papadimitriou, C.: Worst-case equilibria. In: 16th Symposium on Theoretical Aspects in Computer Science, Trier, Germany. LNCS, vol. 1563, pp. 404–413. Springer (1999)
8. Kumar, V.S.A., Marathe, M.V.: Improved results for stackelberg scheduling strategies. In: 29th International Colloquium, Au-

tomata, Languages and Programming. LNCS, pp. 776–787. Springer (2002)
9. Roughgarden, T.: Designing networks for selfish users is hard. In: 42nd IEEE Annual Symposium of Foundations of Computer Science, pp. 472–481 (2001)
10. Roughgarden, T.: Stackelberg scheduling strategies. In: 33rd ACM Annual Symposium on Theory of Computing, pp. 104–113 (2001)
11. Roughgarden, T.: Selfish Routing. Dissertation, Cornell University, USA, May 2002, http://theory.stanford.edu/~tim/
12. Roughgarden, T.: Selfish Routing and the Price of Anarchy. The MIT Press, Cambridge (2005)
13. Roughgarden, T., Tardos, E.: How bad is selfish routing? In: 41st IEEE Annual Symposium of Foundations of Computer Science, pp. 93–102. J. ACM 49(2), pp 236–259, 2002, ACM, New York (2000)
14. Swamy, C.: The effectiveness of stackelberg strategies and tolls for network congestion games. In: ACM-SIAM Symposium on Discrete Algorithms, Philadelphia, PA, USA (2007)
15. von Stackelberg, H.: Marktform und Gleichgewicht. Springer, Vienna (1934)

Statistical Data Compression

► Arithmetic Coding for Data Compression

Statistical Multiple Alignment

2003; Hein, Jensen, Pedersen

István Miklós
Department of Plant Taxonomy and Ecology,
Eötvös Lóránd University, Budapest, Hungary

Keywords and Synonyms

Evolutionary hidden Markov models

Problem Definition

The three main types of mutations modifying biological sequences are insertions, deletions and substitutions. The simplest model involving these three types of mutations is the so-called Thorne–Kishino–Felsenstein model [13]. In this model, the characters of a sequence evolve independently. Each character in the sequence can be substituted with another character according to a prescribed reversible time-continuous Markov model on the possible characters. Insertion-deletions are modeled as a birth-death process, characters evolve independently and identically, with insertion and deletion rates λ and μ.

The multiple statistical alignment problem is to calculate the likelihood of a set of sequences, namely, what is the probability of observing a set of sequences, given all the necessary parameters that describe the evolution of sequences. Hein, Jensen and Pedersen were the first who gave an algorithm to calculate this probability [4]. Their algorithm has $O(5^n L^n)$ running time, where n is the number of sequences, and L is the geometric mean of the sequences. The running time has been improved to $O(2^n L^n)$ by Lunter et al. [10].

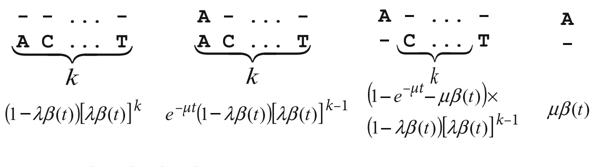
Notations

Insertions and Deletions In the Thorne–Kishino–Felsenstein model (TKF91 model) [13], both the birth and the death processes are Poisson processes with parameters λ and μ, respectively. Since each character evolves independently, the probability of an insertion-deletion pattern given by an alignment can be calculated as the product of the probabilities of patterns. Each pattern starts with an ancestral character, except the first that starts with the beginning of the alignment, end ends before the next ancestral character, except the last that ends at the end of the alignment. The probability of the possible patterns can be found on Fig. 1.

Evolutionary Trees An evolutionary tree is a leaf-labeled, edge weighted, rooted binary tree. Labels are the species related by the evolutionary tree, weights are evolutionary distances. It might happen that the evolutionary changes had different speed at different lineages, and hence the tree is not necessary ultrametric, namely, the root not necessary has the same distance to all leaves.

Given a set S of l-long sequences over alphabet Σ, a substitution model M on Σ and an evolutionary tree T labeled by the sequences. The likelihood of the tree is the probability of observing the sequences at the leaves of the tree, given that the substitution process starts at the root of the tree with the equilibrium distribution. This likelihood is denoted by $P(S|T, M)$. The substitution likelihood problem is to calculate the likelihood of the tree.

Let Σ be a finite alphabet and let $S_1 = s_{1,1}s_{1,2}\ldots s_{1,L_1}$, $S_2 = s_{2,1}s_{2,2}\ldots s_{2,L_2}$, … $S_n = s_{n,1}s_{n,2}\ldots s_{n,L_n}$ be se-


- - ... -
A C ... T
k
$(1-\lambda\beta(t))[\lambda\beta(t)]^k$
A - ... -
A C ... T
k
$e^{-\mu t}(1-\lambda\beta(t))[\lambda\beta(t)]^{k-1}$
A - ... -
- C ... T
k
$(1-e^{-\mu t}-\mu\beta(t))\times(1-\lambda\beta(t))[\lambda\beta(t)]^{k-1}$
A
-
$\mu\beta(t)$

Statistical Multiple Alignment, Figure 1
The probabilities of alignment patterns. From *left* to *right*: k insertions at the beginning of the alignment, a match followed by $k-1$ insertions, a deletion followed by k insertions, a deletion not followed by insertions. $\beta = \frac{1-e^{(\lambda-\mu)t}}{\mu-\lambda e^{(\lambda-\mu)t}}$

quences over this alphabet. Let a TKF91 model $TKF91$ be given with its parameters: substitution model M, insertion rate λ and deletion rate μ. Let T be an evolutionary tree labeled by $S_1, S_2 \ldots S_n$. The multiple statistical alignment problem is to calculate the likelihood of the tree, $P(S_1, S_2, \ldots S_n | T, TKF91)$, given that the TKF91 process starts at the root with the equilibrium distribution.

Multiple Hidden Markov Models It will turn out that the TKF91 model can be transformed to a multiple Hidden Markov Model, therefore it is formally defined here. A multiple Hidden Markov Model (multiple-HMM) is a directed graph with a distinguished Start and End state, (the in-degree of the Start and the out-degree of the End state are both 0), together with the following described transition and emission distributions. Each vertex has a transition distribution over its out-edges. The vertexes can be divided for two classes, the emitting and silent states. Each emitting state emits one-one random character to a prescribed set of sequences, it is possible that a state emits only one character to one sequence. For each state, an emission distribution over the alphabet and the set of sequences gives the probabilities which characters will be emitted to which sequences. The Markov process is a random walk from the Start to the End, following the transition distribution on the out edges. When the walk is in an emitting state, characters are emitted according to the emission distribution of the state. The process is hidden since the observer sees only the emitted sequences, and the observer does not observe which character is emitted by which state, even the observer does not see which characters are co-emitted. The multiple-HMM problem is to calculate the emission probability of a set of sequences for a multiple-HMM. This probability can be calculated with the Forward algorithm that has $O(V^2L^n)$ running time, where V is the number of emitting states in the multiple-HMM, L is the geometric mean of the sequences and n is the number of sequences [2].

Key Results

Substitutions have been modeled with time-continuous Markov models since the late sixties [7] and an efficient algorithm for likelihood calculations was published in 1981 [3]. The running time of this efficient algorithm grows linearly both with the number of sequences and with the length of the sequences being analyzed, and it grows squarely with the size of the alphabet. The algorithm belongs to the class of dynamic programming algorithms.

Thorne, Kishino and Felsenstein gave an $O(nm)$ running time algorithm for calculating the likelihood of an n-long and an m-long sequence under their model [13]. It was not clear for long time how to extend this algorithm to more than two sequences. In 2001, several researchers [6,11] realized that the TKF91 model for two sequences is equivalent with a pair Hidden Markov Model (pair-HMM) in the sense that the transition and emission probabilities of the pair-HMM can be parameterized with λ, μ, and the transition and equilibrium probabilities of the substitution model, moreover there is a bijection between the paths emitting the two sequences and alignments such that the probability of a path in the pair-HMM equals to the probability of the corresponding alignment of the two sequences. Hence the likelihood of two sequences can be calculated with the Forward algorithm of the pair-HMM.

After this discovery, it was relatively easy to develop an algorithm for multiple statistical alignment [4]. The key observation is that a multiple-HMM can be created as a composition of pair-HMMs along the evolutionary tree. This technique was already known in the speech recognition literature [12], and was also rediscovered by Ian Holmes [5], who named this technique as transducer composition. The number of states in the so-created multiple-HMM is $O(5^{\frac{n}{2}})$, where n is the number of leaves of the tree. The emission probabilities are the substitution likelihoods on the tree, which can be efficiently calculated using Felsenstein's algorithm [3]. The running time of the Forward algorithm is $5^n L^n$, where L is the geometric mean of the sequence lengths.

Lunter et al. [10] introduced an algorithm that does not need a multiple-HMM description of the TKF91 model to calculate the likelihood of a tree. Using a logical sieve algorithm, they were able to reduce the running time to $O(2^n L^n)$. They called their algorithm the "one-state recursion" since their dynamic programming algorithm does not need different state of a multiple-HMM to calculate the likelihood correctly.

Applications

Since the running time of the best known algorithm for multiple statistical alignment grows exponentially with the number of sequences, on its own it is not useful in practice. However, Lunter et al. also showed that there is a one-state recursion to calculate the likelihood of the tree given an alignment [8]. The running time of this algorithm grows only linearly with both the alignment length and the number of sequences. Since the number of states in a multiple-HMM that can emit the same multiple alignment column might grow exponentially, this version of the one-state recursion is a significant improvement. The one-state recursion for multiple alignments is used in a Bayesian Markov chain Monte Carlo where the state space is the Descart product of the possible multiple alignments and evolutionary trees. The one-state recursion provides an efficient likelihood calculation for a point in the state space [9].

Csűrös and Miklós introduced a model for gene content evolution that is equivalent with the multiple statistical alignment problem for alphabet size 1 [1]. They gave a polynomial running time algorithm that calculates the likelihood of the tree. The running time is $O(n + hL^2)$, where n is the number of sequences, h is the height of the evolutionary tree, and L is the sum of the sequences lengths.

Open Problems

It is conjectured that the multiple statistical alignment problem cannot be solved in polynomial time for any nontrivial alphabet size. One also can ask what the most likely multiple alignment is or equivalently, what the most probable path in the multiple-HMM is that emits the given sequences. For a set of sequences, a TKF91 model and an evolutionary tree, the decision problem "Is there a multiple alignment that is more probable than p" is conjectured to be NP-complete.

Thorne, Kishino and Felsenstein also introduced a fragment model, also called the TKF92 model, in which multiple insertions and deletions are allowed. The birth process is still a Poisson process, but instead of single characters, fragments of characters are inserted with a geometrically distributed length. The fragments are unbreakable, and the death process is going on the fragments. The TKF92 model for a pair of sequences also can be described into a pair-HMM and the TKF92 model on a tree can be transformed to a multiple-HMM. It is conjectured that there is no one-state recursion for the TKF92 model.

Cross References

► Efficient Methods for Multiple Sequence Alignment with Guaranteed Error Bounds
► Local Alignment (with Affine Gap Weights)

Recommended Reading

1. Csűrös, M., Miklós, I.: A probabilistic model for gene content evolution with duplication, loss, and horizontal transfer. In: Lecture Notes in Bioinformatics, Proceedings of RECOMB2006, vol. 3909, pp. 206–220 (2006)
2. Durbin, R., Eddy, S., Krogh, A., Mitchison, G.: Biological sequence analysis. Cambridge University Press, Cambridge, UK (1998)
3. Felsenstein, J.: Evolutionary trees from DNA sequences: a maximum likelihood approach. J. Mol. Evol. **17**, 368–376 (1981)
4. Hein, J., Jensen, J., Pedersen, C.: Recursions for statistical multiple alignment. PNAS **100**, 14,960–14,965 (2003)
5. Holmes, I.: Using guide trees to construct multiple-sequence evolutionary hmms. Bioinform. **19**, i147–i157 (2003)
6. Holmes, I., Bruno, W.J.: Evolutionary HMMs: a Bayesian approach to multiple alignment. Bioinform. **17**(9), 803–820 (2001)
7. Jukes, T.H., Cantor, C.R.: Evolution of protein molecules. In: Munro (ed.) Mammalian Protein Metabolism, pp. 21–132. Acad. Press (1969)
8. Lunter, G., Miklós, I., Drummond, A., Jensen, J., Hein, J.: Bayesian phylogenetic inference under a statistical indel model. In: Lecture Notes in Bioinformatics, Proceedings of WABI2003, vol. 2812, pp. 228–244 (2003)
9. Lunter, G., Miklós, I., Drummond, A., Jensen, J., Hein, J.: Bayesian coestimation of phylogeny and sequence alignment. BMC Bioinformatics (2005)
10. Lunter, G.A., Miklós, I., Song, Y.S., Hein, J.: An efficient algorithm for statistical multiple alignment on arbitrary phylogenetic trees. J. Comp. Biol. **10**(6), 869–889 (2003)
11. Metzler, D., Fleißner, R., Wakolbringer, A., von Haeseler, A.: Assessing variability by joint sampling of alignments and mutation rates. J. Mol. Evol. **53**, 660–669 (2001)
12. Pereira, F., Riley, M.: Speech recognition by composition of weighted finite automata. In: Finite-State Language Processing, pp. 149–173. MIT Press, Cambridge (1997)
13. Thorne, J.L., Kishino, H., Felsenstein, J.: An evolutionary model for maximum likelihood alignment of DNA sequences. J. Mol. Evol. **33**, 114–124 (1991)

Statistical Query Learning

1998; Kearns

VITALY FELDMAN
Department of Engineering and Applied Sciences,
Harvard University, Cambridge, MA, USA

Problem Definition

The problem deals with learning $\{-1, +1\}$-valued functions from random labeled examples in the presence of random noise in the labels. In the *random classification noise* model of Angluin and Laird [1] the label of each example given to the learning algorithm is flipped randomly and independently with some fixed probability η called the *noise rate*. The model is the extension of Valiant's PAC model [14] that formalizes the simplest type of white label noise.

Robustness to this relatively benign noise is an important goal in the design of learning algorithms. Kearns defined a powerful and convenient framework for constructing noise-tolerant algorithms based on *statistical queries*. Statistical query (SQ) learning is a natural restriction of PAC learning that models algorithms that use statistical properties of a data set rather than individual examples.

Kearns demonstrated that any learning algorithm that is based on statistical queries can be automatically converted to a learning algorithm in the presence of random classification noise of arbitrary rate smaller than the information-theoretic barrier of 1/2. This result was used to give the first noise-tolerant algorithm for a number of important learning problems. In fact, virtually all known noise-tolerant PAC algorithms were either obtained from SQ algorithms or can be easily cast into the SQ model.

Definitions and Notation

Let C be a class of $\{-1, +1\}$-valued functions (also called *concepts*) over an input space X. In the basic PAC model a learning algorithm is given examples of an unknown function f from C on points randomly chosen from some unknown distribution $\mathcal{D}$ over X and should produce a hypothesis h that approximates f. More formally, an *example oracle* $\mathrm{EX}(f, \mathcal{D})$ is an oracle that upon being invoked returns an example $\langle x, f(x)\rangle$, where x is chosen randomly with respect to $\mathcal{D}$, independently of any previous examples. A learning algorithm for C is an algorithm that for every $\varepsilon > 0$, $\delta > 0$, $f \in C$, and distribution $\mathcal{D}$ over X, given ε, δ, and access to $\mathrm{EX}(f, \mathcal{D})$ outputs, with probability at least $1 - \delta$, a hypothesis h that ε-approximates f with respect to $\mathcal{D}$ (i. e. $\mathbf{Pr}_{\mathcal{D}}[f(x) \neq h(x)] \leq \varepsilon$). Efficient learning algorithms are algorithms that run in time polynomial in $1/\varepsilon$, $1/\delta$, and the size of the learning problem s. The size of a learning problem is determined by the description length of f under some fixed representation scheme for functions in C and the description length of an element in X (often proportional to the dimension n of the input space).

A number of variants of this basic framework are commonly considered. The basic PAC model is also referred to as *distribution-independent* learning to distinguish it from *distribution-specific* PAC learning in which the learning algorithm is required to learn with respect to a single distribution $\mathcal{D}$ known in advance. A *weak* learning algorithm is a learning algorithm that can produce a hypothesis whose error on the target concept is noticeably less than 1/2 (and not necessarily any $\varepsilon > 0$). More precisely, a weak learning algorithm produces a hypothesis h such that $\mathbf{Pr}_{\mathcal{D}}[f(x) \neq h(x)] \leq 1/2 - 1/p(s)$ for some fixed polynomial p. The basic PAC model is often referred to as *strong* learning in this context.

In the random classification noise model $\mathrm{EX}(f, \mathcal{D})$ is replaced by a faulty oracle $\mathrm{EX}^{\eta}(f, \mathcal{D})$, where η is the noise rate. When queried, this oracle returns a noisy example $\langle x, b\rangle$ where $b = f(x)$ with probability $1 - \eta$ and $\neg f(x)$ with probability η independently of previous examples. When η approaches 1/2 the label of the corrupted example approaches the result of a random coin flip, and therefore the running time of learning algorithms in this model is allowed to depend on $\frac{1}{1-2\eta}$ (the dependence must be polynomial for the algorithm to be considered efficient). For simplicity one usually assumes that η is known to the learning algorithm. This assumption can be removed using a simple technique due to Laird [12].

To formalize the idea of learning from statistical properties of a large number of examples, Kearns introduced a new oracle $\mathrm{STAT}(f, \mathcal{D})$ that replaces $\mathrm{EX}(f, \mathcal{D})$. The oracle $\mathrm{STAT}(f, \mathcal{D})$ takes as input a *statistical query* (SQ) of the form (χ, τ) where χ is a $\{-1, +1\}$-valued function on labeled examples and $\tau \in [0, 1]$ is the *tolerance* parameter. Given such a query the oracle responds with an estimate of $\mathbf{Pr}_{\mathcal{D}}[\chi(x, f(x)) = 1]$ that is accurate to within an additive $\pm\tau$. Chernoff bounds easily imply that $\mathrm{STAT}(f, \mathcal{D})$ can, with high probability, be simulated using $\mathrm{EX}(f, \mathcal{D})$ by estimating $\mathbf{Pr}_{\mathcal{D}}[\chi(x, f(x)) = 1]$ on $O(\tau^{-2})$ examples. Therefore the SQ model is a restriction of the PAC model. Efficient SQ algorithms allow only efficiently evaluable χ's and impose an inverse polynomial lower bound on the tolerance parameter over all oracle calls.

Key Results

Statistical Queries and Noise-Tolerance

The main result given by Kearns is a way to simulate statistical queries using noisy examples.

Lemma 1 ([10]) *Let (χ, τ) be a statistical query such that χ can be evaluated on any input in time T and let $EX^{\eta}(f, \mathcal{D})$ be a noisy oracle. The value $\mathbf{Pr}_{\mathcal{D}}[\chi(x, f(x)) = 1]$ can, with probability at least $1 - \delta$, be estimated within τ using $O(\tau^{-2}(1-2\eta)^{-2} \log(1/\delta))$ examples from $EX^{\eta}(f, \mathcal{D})$ and time $O(\tau^{-2}(1 - 2\eta)^{-2} \log(1/\delta) \cdot T)$.*

This simulation is based on estimating several probabilities using examples from the noisy oracle and then offsetting the effect of noise. The lemma implies that any efficient SQ algorithm for a concept class C can be converted to an efficient learning algorithm for C tolerating random classification noise of any rate $\eta < 1/2$.

Theorem 2 ([10]) *Let C be a concept class efficiently PAC learnable from statistical queries. Then C is efficiently PAC learnable in the presence of random classification noise of rate η for any $\eta < 1/2$.*

Kearns also shows that in order to simulate all the statistical queries used by an algorithm one does not necessarily need new examples for each estimation. Instead, assuming that the set of possible queries of the algorithm has Vapnik–Chervonenkis dimension d, all its statistical queries can be simulated using $\tilde{O}(d\tau^{-2}(1-2\eta)^{-2}\log(1/\delta)+\varepsilon^{-2})$ examples [10].

One of the most significant results on learning in the distribution-independent PAC learning model is the equivalence of weak and strong learnability demonstrated by Schapire's celebrated *boosting* method [13]. Aslam and Decatur showed that this equivalence holds in the SQ model as well [2].

A natural way to extend the SQ model is to allow query functions that depend on a t-tuple of examples instead of just one example. Blum et al. proved that this extension does not increase the power of the model as long as $t = O(\log s)$ [5].

Statistical Query Dimension

The restricted way in which SQ algorithms use examples makes it simpler to understand the limitations of efficient learning in this model. A long-standing open problem in learning theory is learning of the concept class of all parity functions over $\{0,1\}^n$ with noise (a parity function is a XOR of some subset of n Boolean inputs). Kearns has demonstrated that parities cannot be efficiently learned using statistical queries even under the uniform distribution over $\{0,1\}^n$ [10]. This hardness result is unconditional in the sense that it does not rely on any unproven complexity assumptions.

The technique of Kearns was generalized by Blum et al. who proved that efficient SQ learnability of a concept class C is characterized by a relatively simple combinatorial parameter of C called the *statistical query dimension* [4]. The quantity they defined measures the maximum number of "nearly uncorrelated" functions in a concept class. More formally,

Definition 3 For a concept class C and distribution $\mathcal{D}$, the *statistical query dimension* of C with respect to $\mathcal{D}$, denoted SQ-DIM$(C,\mathcal{D})$, is the largest number d such that C contains d functions $f_1, f_2, \ldots, f_d$ such that for all $i \neq j$, $|\mathbf{E}_{\mathcal{D}}[f_i f_j]| \leq \frac{1}{d^3}$.

Blum et al. relate the SQ dimension to learning in the SQ model as follows.

Theorem 4 ([4]) *Let C be a concept class and $\mathcal{D}$ be a distribution such that SQ-DIM$(C,\mathcal{D}) = d$.*

- *If all queries are made with tolerance of at least $1/d^{1/3}$, then at least $d^{1/3}$ queries are required to learn C with error $1/2 - 1/d^3$ in the SQ model.*
- *There exists an algorithm for learning C with respect to $\mathcal{D}$ that makes d fixed queries, each of tolerance $1/3d^3$, and finds a hypothesis with error at most $1/2 - 1/3d^3$.*

Thus SQ-DIM characterizes weak learnability in the SQ model up to a polynomial factor. Parity functions are uncorrelated with respect to the uniform distribution and therefore any concept class that contains a superpolynomial number of parity functions cannot be learned by statistical queries with respect to the uniform distribution. This for example includes such important concept classes as *k-juntas* over $\{0,1\}^n$ (or functions that depend on at most k input variables) for $k = \omega(1)$ and *decision trees* of superconstant size.

The following important result is due to Blum et al. [5]:

Theorem 5 ([5]) *For any constant $\eta < 1/2$, parities that depend on the first $\log n \log\log n$ input variables are efficiently PAC learnable in the presence of random classification noise of rate η.*

Since there are $n^{\log\log n}$ parity functions that depend on the first $\log n \log\log n$ input variables, this shows that there exist concept classes that are efficiently learnable in the presence of noise (at constant rate $\eta < 1/2$) but are not efficiently learnable in the SQ model.

Applications

Learning by statistical queries was used to obtain noise-tolerant algorithms for a number of important concept classes. One of the ways this can be done is by showing that a PAC learning algorithm can be modified to use statistical queries instead of random examples. Examples of learning problems for which the first noise-tolerant algorithm was obtained using this approach include [10]:

- Learning decision trees of constant rank.
- *Attribute-efficient* algorithms for learning conjunctions.
- Learning axis-aligned rectangles over $\mathbb{R}^n$.
- Learning AC^0 (constant-depth unbounded fan-in) Boolean circuits over $\{0,1\}^n$ with respect to the uniform distribution in quasipolynomial time.

Blum et al. also use the SQ model to show that their algorithm for learning linear threshold functions is noise-tolerant [3], resolving an important open problem.

The ideas behind the use of statistical queries to produce noise tolerant algorithms were adapted to learning using *membership queries* (or ability to ask for the value of the unknown function at any point). There the noise model has to be modified slightly to prevent the learner from asking for independently corrupted labels on the same point. An appropriate modification is the introduction of *persistent classification noise* by Goldman et al. [7]. In this model, as before, the answer to a query at each point x is flipped with probability $1-\eta$. However, if the membership oracle was already queried about the value of f at some specific point x or x was already generated as a random example, the returned label has the same value as in the first occurrence.

Extensions of the SQ model suggested by Jackson et al. [9] and Bshouty and Feldman [6] allow any algorithm based on these extended statistical queries to be converted to a noise-tolerant PAC algorithm with membership queries. In particular, they used this approach to convert Jackson's algorithm for learning DNF with respect to the uniform distribution to a noise-tolerant one. Bshouty and Feldman also show that learnability in their extension can be characterized using a dimension similar to the SQ dimension of Blum et al. [4].

Open Problems

The main questions related to learning with random classification noise are still open. Is every concept class efficiently learnable in the PAC model also learnable in the presence of random classification noise? Is every concept class efficiently learnable in the presence of random classification noise of arbitrarily high rate (less than 1/2) also efficiently learnable using statistical queries? Note that the algorithm of Blum et al. assumes that the noise rate is a constant and therefore does not provide a complete answer to this question [5]. For both questions a central issue seems to be obtaining a better understanding of the complexity of learning parities with noise.

Another important direction of research is learning with weaker assumptions on the nature of noise. A natural model that places no assumptions on the way in which the labels are corrupted is the *agnostic* learning model defined by Haussler [8] and Kearns et al. [11]. Efficient learning algorithms that can cope with this, possibly adversarial, noise is a very desirable if hard to achieve goal. For example, learning conjunctions of input variables in this model is an open problem known to be at least as hard as learning DNF expressions in the PAC model [11]. It is therefore important to identify and investigate useful and general models of noise based on less pessimistic assumptions.

Cross References

- ▶ Attribute-Efficient Learning
- ▶ Learning Constant-Depth Circuits
- ▶ Learning DNF Formulas
- ▶ Learning Heavy Fourier Coefficients of Boolean Functions
- ▶ Learning with Malicious Noise
- ▶ PAC Learning

Recommended Reading

1. Angluin, D., Laird, P.: Learning from noisy examples. Mach. Learn. **2**, 343–370 (1988)
2. Aslam, J., Decatur, S.: Specification and simulation of statistical query algorithms for efficiency and noise tolerance. J. Comput. Syst. Sci. **56**, 191–208 (1998)
3. Blum, A., Frieze, A., Kannan, R., Vempala, S.: A polynomial time algorithm for learning noisy linear threshold functions. Algorithmica **22**(1/2), 35–52 (1997)
4. Blum, A., Furst, M., Jackson, J., Kearns, M., Mansour, Y., Rudich, S.: Weakly learning DNF and characterizing statistical query learning using Fourier analysis. In: Proceedings of STOC, pp. 253–262 (1994)
5. Blum, A., Kalai, A., Wasserman, H.: Noise-tolerant learning, the parity problem, and the statistical query model. J. ACM **50**(4), 506–519 (2003)
6. Bshouty, N., Feldman, V.: On using extended statistical queries to avoid membership queries. J. Mach. Learn. Res. **2**, 359–395 (2002)
7. Goldman, S., Kearns, M., Schapire, R.: Exact identification of read-once formulas using fixed points of amplification functions. SIAM J. Comput. **22**(4), 705–726 (1993)
8. Haussler, D.: Decision theoretic generalizations of the PAC model for neural net and other learning applications. Inf. Comput. **100**(1), 78–150 (1992)
9. Jackson, J., Shamir, E., Shwartzman, C.: Learning with queries corrupted by classification noise. In: Proceedings of the Fifth Israel Symposium on the Theory of Computing Systems, pp. 45–53 (1997)
10. Kearns, M.: Efficient noise-tolerant learning from statistical queries. J. ACM **45**(6), 983–1006 (1998)
11. Kearns, M., Schapire, R., Sellie, L.: Toward efficient agnostic learning. Mach. Learn. **17**(2-3), 115–141 (1994)
12. Laird, P.: Learning from good and bad data. Kluwer Academic Publishers (1988)
13. Schapire, R.: The strength of weak learnability. Mach. Learn. **5**(2), 197–227 (1990)
14. Valiant, L.: A theory of the learnable. Commun. ACM **27**(11), 1134–1142 (1984)

Steiner Forest

1995; Agrawal, Klein, Ravi

Guido Schäfer
Institute for Mathematics and Computer Science, Technical University of Berlin, Berlin, Germany

Keywords and Synonyms

Requirement join; R-join, Requirement Join

Problem Definition

The *Steiner forest problem* is a fundamental problem in network design. Informally, the goal is to establish connections between pairs of vertices in a given network at minimum cost. The problem generalizes the well-known *Steiner tree problem*. As an example, assume that a telecommunication company receives communication requests from their customers. Each customer asks for a connection between two vertices in a given network. The company's goal is to build a minimum cost network infrastructure such that all communication requests are satisfied.

Formal Definition and Notation

More formally, an instance $I = (G, c, R)$ of the Steiner forest problem is given by an undirected graph $G = (V, E)$ with vertex set V and edge set E, a non-negative cost function $c\colon E \to \mathbb{Q}^+$, and a set of vertex pairs $R = \{(s_1, t_1), \dots, (s_k, t_k)\} \subseteq V \times V$. The pairs in R are called *terminal pairs*. A feasible solution is a subset $F \subseteq E$ of the edges of G such that for every terminal pair $(s_i, t_i) \in R$ there is a path between s_i and t_i in the subgraph $G[F]$ induced by F. Let the cost $c(F)$ of F be defined as the total cost of all edges in F, i. e., $c(F) = \sum_{e\in F} c(e)$. The goal is to find a feasible solution F of minimum cost $c(F)$. It is easy to see that there exists an optimum solution which is a forest.

The Steiner forest problem may alternatively be defined by a set of *terminal groups* $R = \{g_1, \dots, g_k\}$ with $g_i \subseteq V$ instead of terminal pairs. The objective is to compute a minimum cost subgraph such that all terminals belonging to the same group are connected. This definition is equivalent to the one given above.

Related Problems

A special case of the Steiner forest problem is the *Steiner tree problem* (see also the entry ► Steiner Tree). Here, all terminal pairs share a common root vertex $r \in V$, i. e., $r \in \{s_i, t_i\}$ for all terminal pairs $(s_i, t_i) \in R$. In other words, the problem consists of a set of terminal vertices $R \subseteq V$ and a root vertex $r \in V$ and the goal is to connect the terminals in R to r in the cheapest possible way. A minimum cost solution is a tree.

The *generalized Steiner network problem* (see the entry ► Generalized Steiner Network), also known as the *survivable network design problem*, is a generalization of the Steiner forest problem. Here, a *connectivity requirement* function $r\colon V \times V \to \mathbb{N}$ specifies the number of edge disjoint paths that need to be established between every pair of vertices. That is, the goal is to find a minimum cost multi-subset H of the edges of G (H may contain the same edge several times) such that for every pair of vertices $(x, y) \in V$ there are $r(x, y)$ edge disjoint paths from x to y in $G[H]$. The goal is to find a set H of minimum cost. Clearly, if $r(x, y) \in \{0, 1\}$ for all $(x, y) \in V \times V$, this problem reduces to the Steiner forest problem.

Key Results

Agrawal, Klein and Ravi [1,2] give an approximation algorithm for the Steiner forest problem that achieves an approximation ratio of 2. More precisely, the authors prove the following theorem.

Theorem 1 *There exists an approximation algorithm that for every instance $I = (G, c, R)$ of the Steiner forest problem, computes a feasible forest F such that*

$$c(F) \le \left(2 - \frac{1}{k}\right) \cdot \mathit{OPT}(I) ,$$

where k is the number of terminal pairs in R and $\mathit{OPT}(I)$ is the cost of an optimal Steiner forest for I.

Related Work

The Steiner tree problem is NP-hard [10] and APX-complete [4,8]. The current best lower bound on the achievable approximation ratio for the Steiner tree problem is 1.0074 [21]. Goemans and Williamson [11] generalized the results obtained by Agrawal, Klein and Ravi to a larger class of connectivity problems, which they term *constrained forest problems*. For the Steiner forest problem, their algorithm achieves the same approximation ratio of $(2 - 1/k)$. The algorithms of Agrawal, Klein and Ravi [2] and Goemans and Williamson [11] are both based on the classical *undirected cut formulation* for the Steiner forest problem [3]. The integrality gap of this relaxation is known to be $(2 - 1/k)$ and the results in [2,11] are therefore tight. Jain [15] presents a 2-approximation algorithm for the generalized Steiner network problem.

Primal-Dual Algorithm

The main ideas of the algorithm by Agrawal, Klein and Ravi [2] are sketched below; subsequently, AKR is used to refer to this algorithm. The description given here differs from the one in [2]; the interested reader is referred to [2] for more details.

The algorithm is based on the following integer programming formulation for the Steiner forest problem. Let $I = (G, c, R)$ be an instance of the Steiner forest problem. Associate an indicator variable $x_e \in \{0, 1\}$ with every edge $e \in E$. The value of x_e is 1 if e is part of the forest F and 0 otherwise. A subset $S \subseteq V$ of the vertices is called a *Steiner cut* if there exists at least one terminal pair $(s_i, t_i) \in R$ such that $|\{s_i, t_i\} \cap S| = 1$; S is said to *separate* terminal pair (s_i, t_i). Let $\mathcal{S}$ be the set of all Steiner cuts. For a subset $S \subseteq V$, define $\delta(S)$ as the the set of all edges in E that have exactly one endpoint in S. Given a Steiner cut $S \in \mathcal{S}$, any feasible solution F of I must contain at least one edge that *crosses* the cut S, i. e., $\sum_{e \in \delta(S)} x_e \geq 1$. This gives rise to the following *undirected cut formulation*:

$$\begin{aligned} &\text{minimize} && \sum_{e \in E} c(e) x_e && \text{(IP)} \\ &\text{subject to} && \sum_{e \in \delta(S)} x_e \geq 1 \quad \forall S \in \mathcal{S} && (1) \\ & && x_e \in \{0, 1\} \quad \forall e \in E\,. && (2) \end{aligned}$$

The dual of the linear programming relaxation of (IP) has a variable y_S for every Steiner cut $S \in \mathcal{S}$. There is a constraint for every edge $e \in E$ that requires that the total dual assigned to sets $S \in \mathcal{S}$ that contain exactly one endpoint of e is at most the cost $c(e)$ of the edge:

$$\begin{aligned} &\text{maximize} && \sum_{S \in \mathcal{S}} y_S && \text{(D)} \\ &\text{subject to} && \sum_{S \in \mathcal{S}:\ e \in \delta(S)} y_S \leq c(e) \quad \forall e \in E && (3) \\ & && y_S \geq 0 \quad \forall S \in \mathcal{S}\,. && (4) \end{aligned}$$

Algorithm AKR is based on the *primal-dual schema* (see, e. g., [22]). That is, the algorithm constructs both a feasible primal solution for (IP) and a feasible dual solution for (D). The algorithm starts with an infeasible primal solution and reduces its degree of infeasibility as it progresses. At the same time, it creates a feasible dual packing of subsets of large total value by raising dual variables of Steiner cuts.

One can think of an execution of AKR as a process over time. Let x^τ and y^τ, respectively, be the primal incidence vector and feasible dual solution at time τ. Initially, let $x_e^0 = 0$ for all $e \in E$ and $y_S^0 = 0$ for all $S \in \mathcal{S}$. Let F^τ denote the forest corresponding to the set of edges with $x_e^\tau = 1$. A tree T in F^τ is called *active* at time τ if it contains a terminal that is separated from its mate; otherwise it is *inactive*. Intuitively, AKR grows trees in F^τ that are active. At the same time, the algorithm raises dual values of Steiner cuts that correspond to active trees. If two active trees collide, they are merged. The process terminates if all trees are inactive and thus there are no unconnected terminal pairs. The interplay of the primal (growing trees) and the dual process (raising duals) is somewhat subtle and outlined next.

An edge $e \in E$ is *tight* if the corresponding constraint (3) holds with equality; a path is tight if all its edges are tight. Let H^τ be the subgraph of G that is induced by the tight edges for dual y^τ. The connected components of H^τ induce a partition $\mathcal{C}^\tau$ on the vertex set V. Let $\mathcal{S}^\tau$ be the set of all Steiner cuts contained in $\mathcal{C}^\tau$, i. e., $\mathcal{S}^\tau = \mathcal{C}^\tau \cap \mathcal{S}$. AKR raises the dual values y_S for all sets $S \in \mathcal{S}^\tau$ uniformly at all times $\tau \geq 0$. Note that y^τ is dual feasible. The algorithm maintains the invariant that F^τ is a subgraph of H^τ at all times. Consider the event that a path P between two trees T_1 and T_2 of F^τ becomes tight. The missing edges of P are then added to F^τ and the process continues. Eventually, all trees in F^τ are inactive and the process halts.

Applications

The computation of (approximate) solutions for the Steiner forest problem has various applications both in theory and practice; only a few recent developments are mentioned here.

Algorithms for more complex network design problems often rely on good approximation algorithms for the Steiner forest problem. For example, the recent approximation algorithms [6,9,12] for the *multi-commodity rent-or-buy problem* (MRoB) are based on the random sampling framework by Gupta et al. [12,13]. The framework uses a Steiner forest approximation algorithm that satisfies a certain *strictness* property as a subroutine. Fleischer et al. [9] show that AKR meets this strictness requirement, which leads to the current best 5-approximation algorithm for MRoB. The strictness property also plays a crucial role in the boosted sampling framework by Gupta et al. [14] for two-stage stochastic optimization problems with recourse.

Online versions of Steiner tree and forest problems have been studied by by Awerbuch et al. [5] and Berman and Coulston [7]. In the area of algorithmic game theory, the development of *group-strategyproof cost sharing mechanisms* for network design problems such as the Steiner tree problem has recently received a lot of attention; see e. g., [16,17,19,20]. An adaptation of AKR yields such a cost sharing mechanism for the Steiner forest problem [18].

Cross References

► Generalized Steiner Network
► Steiner Trees

Recommended Reading

The interested reader is referred in particular to the articles [2,11] for a more detailed description of primal-dual approximation algorithms for general network design problems.

1. Agrawal, A., Klein, P., Ravi, R.: When trees collide: an approximation algorithm for the generalized Steiner problem on networks. In: Proc. of the 23rd Annual ACM Symposium on Theory of Computing, Association for Computing Machinery, New York, pp. 134–144 (1991)
2. Agrawal, A., Klein, P., Ravi, R.: When trees collide: An approximation algorithm for the generalized Steiner problem in networks. SIAM J. Comput. **24**(3), 445–456 (1995)
3. Aneja, Y.P.: An integer linear programming approach to the Steiner problem in graphs. Networks **10**(2), 167–178 (1980)
4. Arora, S., Lund, C., Motwani, R., Sudan, M., Szegedy, M.: Proof verification and the hardness of approximation problems. J. ACM **45**(3), 501–555 (1998)
5. Awerbuch, B., Azar, Y., Bartal, Y.: On-line generalized Steiner problem. In: Proc. of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms, Society for Industrial and Applied Mathematics, Philadelphia, 2005, pp. 68–74 (1996)
6. Becchetti, L., Könemann, J., Leonardi, S., Pál, M.: Sharing the cost more efficiently: improved approximation for multicommodity rent-or-buy. In: Proc. of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms, Society for Industrial and Applied Mathematics, Philadelphia, pp. 375–384 (2005)
7. Berman, P., Coulston, C.: On-line algorithms for Steiner tree problems. In: Proc. of the 29th Annual ACM Symposium on Theory of Computing, pp. 344–353. Association for Computing Machinery, New York (1997)
8. Bern, M., Plassmann, P.: The Steiner problem with edge lengths 1 and 2. Inf. Process. Lett. **32**(4), 171–176 (1989)
9. Fleischer, L., Könemann, J., Leonardi, S., Schäfer, G.: Simple cost sharing schemes for multicommodity rent-or-buy and stochastic Steiner tree. In: Proc. of the 38th Annual ACM Symposium on Theory of Computing, pp. 663–670. Association for Computing Machinery, New York (2006)
10. Garey, M.R., Johnson, D.S.: Computers and intractability: a guide to the theory of NP-completeness. Freeman, San Francisco (1979)
11. Goemans, M.X., Williamson, D.P.: A general approximation technique for constrained forest problems. SIAM J. Comput. **24**(2), 296–317 (1995)
12. Gupta, A., Kumar, A., Pál, M., Roughgarden, T.: Approximation via cost-sharing: a simple approximation algorithm for the multicommodity rent-or-buy problem. In: Proc. of the 44th Annual IEEE Symposium on Foundations of Computer Science, pp. 606–617., IEEE Computer Society, Washington (2003)
13. Gupta, A., Kumar, A., Pál, M., Roughgarden, T.: Approximation via cost-sharing: simpler and better approximation algorithms for network design. J. ACM 54(3), Article 11 (2007)
14. Gupta, A., Pál, M., Ravi, R., Sinha, A.: Boosted sampling: approximation algorithms for stochastic optimization. In: Proc. of the 36th Annual ACM Symposium on Theory of Computing, pp. 417–426. Association for Computing Machinery, New York (2004)
15. Jain, K.: A factor 2 approximation for the generalized Steiner network problem. Combinatorica **21**(1), 39–60 (2001)
16. Jain, K., Vazirani, V.V.: Applications of approximation algorithms to cooperative games. In: Proc. of the 33rd Annual ACM Symposium on Theory of Computing, Association for Computing Machinery, New York, pp. 364–372 (2001)
17. Kent, K., Skorin-Kapov, D.: Population monotonic cost allocation on mst's. In: Proc. of the 6th International Conference on Operational Research, Croatian Operational Research Society, Zagreb, pp. 43–48 (1996)
18. Könemann, J., Leonardi, S., Schäfer, G.: A group-strategyproof mechanism for Steiner forests. In: Proc. of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 612–619. Society for Industrial and Applied Mathematics, Philadelphia (2005)
19. Megiddo, N.: Cost allocation for Steiner trees. Networks **8**(1), 1–6 (1978)
20. Moulin, H., Shenker, S.: Strategyproof sharing of submodular costs: budget balance versus efficiency. Econ. Theor. **18**(3), 511–533 (2001)
21. Thimm, M.: On the approximability of the Steiner tree problem. Theor. Comput. Sci. **295**(1–3), 387–402 (2003)
22. Vazirani, V.V.: Approximation algorithms. Springer, Berlin (2001)

Steiner Trees

2006; Du, Graham, Pardalos, Wan, Wu, Zhao

Yaocun Huang, Weili Wu
Department of Computer Science,
University of Texas at Dallas, Richardson, TX, USA

Keywords and Synonyms

Approximation algorithm design

Definition

Given a set of points, called *terminals*, in a metric space, the problem is to find the shortest tree interconnecting all terminals. There are three important metric spaces for Steiner trees, the Euclidean plane, the rectilinear plane, and the edge-weighted network. The Steiner tree problems in those metric spaces are called the *Euclidean Steiner Tree (EST)*, the *Rectilinear Steiner Tree (RST)*, and the *Network Steiner Tree (NST)*, respectively. EST and RST has been found to have polynomial-time approximation schemes (PTAS) by using adaptive partition. However, for NST, there exists a positive number r such that computing r-approximation is NP-hard. So far, the best performance ratio of polynomial-time approximation for NST is achieved by k-restricted Steiner trees. However, in practice, the iterated 1-Steiner tree is used very often. Actually, the iterated 1-Steiner was proposed as a candidate of good approximation for Steiner minimum trees a long time ago. It has a very good record in computer experiments, but no correct analysis was given showing the iterated 1-Steiner tree having a performance ratio better than that of the minimum spanning tree until the recent work by Du et al.[9]. There is minimal difference in construction of the 3-restricted Steiner tree and the iterated 1-Steiner tree, which makes a big difference in analysis of those two types of trees. Why does the difficulty of analysis make so much difference? This will be explained in this article.

History and Background

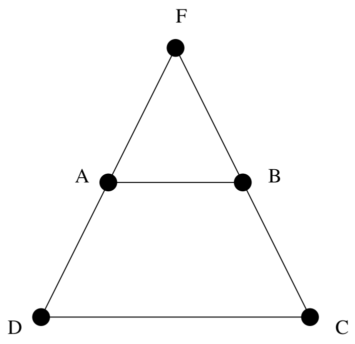
The Steiner tree problem was proposed by Gauss in 1835 as a generalization of the Fermat problem. Given three points A, B, and C in the Euclidean plane, Fermat studied the problem of finding a point S to minimize $|SA| + |SB| + |SC|$. He determined that when all three inner angles of triangle ABC are less than 120°, the optimal S should be at the position that $\angle ASB = \angle BSC = \angle CSA = 120°$.

The generalization of the Fermat problem has two directions:

1. Given n points in the Euclidean plane, find a point S to minimize the total distance from S to n given points. This is still called the Fermat problem.
2. Given n points in the Euclidean plane, find the shortest network interconnecting all given points.

Gauss found the second generalization through communication with Schumacher. On March 19, 1836, Schumacher wrote a letter to Gauss and mentioned a paradox about Fermat's problem: Consider a convex quadrilateral $ABCD$. It is known that the solution of Fermat's problem for four points A, B, C, and D is the intersection E of diagonals AC and BD. Suppose extending DA and CB can obtain an intersection F. Now, move A and B to F. Then E will also be moved to F. However, when the angle at F is less than 120°, the point F cannot be the solution of Fermat's problem for three given points F, D, and C. What happens? (Fig. 1.)

On March 21, 1836, Gauss wrote a letter replying to Schumacher in which he explained that the mistake of Schumacher's paradox occurs at the place where Fermat's problem for four points A, B, C, and D is changed to



Steiner Trees, Figure 1

Fermat's problem for three points F, C, and D. When A and B are identical to F, the total distance from E to four points A, B, C, and D equals $2|EF| + |EC| + |ED|$, not $|EF| + |EC| + |ED|$. Thus, the point E may not be the solution of Fermat's problem for F, C, and D. More importantly, Gauss proposed a new problem. He said that it is more interesting to find the shortest network rather than a point. Gauss also presented several possible connections of the shortest network for four given points.

It was unfortunate that Gauss' letter was not seen by researchers of Steiner trees at an earlier stage. Especially, R. Courant and H. Robbins who in their popular book *What is mathematics?* (published in 1941) [6] called Gauss' problem the Steiner tree so that "Steiner tree" became a popular name for the problem.

The Steiner tree became an important research topic in mathematics and computer science due to its applications in telecommunication and computer networks. Starting with Gilbert and Pollak's work published in 1968, many publications on Steiner trees have been generated to solve various problems concerning it.

One well-known problem is the *Gilbert–Pollak conjecture* on the Steiner ratio, which is the least ratio of lengths between the Steiner minimum tree and the minimum spanning tree on the same set of given points. Gilbert and Pollak in 1968 conjectured that the Steiner ratio in the Euclidean plane is $\sqrt{3}/2$ which is achieved by three vertices of an equilateral triangle. A great deal of research effort has been put into the conjecture and it was finally proved by Du and Hwang [7].

Another important problem is called the *better approximation*. For a long time no approximation could be proved to have a performance ratio smaller than the inverse of the Steiner ratio. Zelikovsky [14] made the first breakthrough. He found a polynomial-time 11/6-approximation for NST which beats 1/2, the inverse of the Steiner ratio in the edge-weighted network. Later, Berman and Ramaiye [2] gave a polynomial-time 92/72-approximation for RST and Du, Zhang, and Feng [8] closed the story by showing that in any metric space, there exists a polynomial-time approximation with a performance ratio better than the inverse of the Steiner ratio provided that for any set of a fixed number of points, the Steiner minimum tree is polynomial-time computable.

All the above better approximations came from the family of k-restricted Steiner trees. By improving some detail of construction, the constant performance ratio was decreasing, but the improvements were also becoming smaller. In 1996, Arora [1] made significant progress for EST and RST. He showed the existence of PTAS for EST and RST. Therefore, the theoretical researchers now pay more attention to NST. Bern and [3] showed that NST is MAX SNP-complete. This means that there exists a positive number r, computing the r-approximation for NST is NP-hard. The best-known performance for NST was given by Robin and Zelikovsky [12]. They also gave a very simple analysis to a well-known heuristic, the iterated 1-Steiner tree for pseudo-bipartite graphs.

Analysis of the iterated 1-Steiner tree is another long-standing open problem. Since Chang [4,5] proposed that the iterated 1-Steiner tree approximates the Steiner minimum tree in 1972, its performance has been claimed to be very good through computer experiments[10,13], but no theoretical analysis supported this claim. Actually, both the k-restricted Steiner tree and the iterated 1-Steiner tree are obtained by greedy algorithms, but with different types of potential functions. For the iterated 1-Steiner tree, the potential function is non-submodular, but for the k-restricted Steiner tree, it is submodular; a property that holds for k-restricted Steiner trees may not hold for iterated 1-Steiner trees. Actually, the submodularity of potential function is very important in analysis of greedy approximations [11]. Du et al. [9] gave a correct analysis for the iterated 1-Steiner tree with a general technique to deal with non-submodular potential function.

Key Results

Consider input edge-weighted graph $G = (V, E)$ of NST. Assume that G is a complete graph and the edge-weight satisfies the triangular inequality, otherwise, consider the complete graph on V with each edge (u, v) having a weight equal to the length of the shortest path between u and v in G. Given a set P of terminals, a *Steiner tree* is a tree interconnecting all given terminals such that every leaf is a terminal.

In a Steiner tree, a terminal may have degree more than one. The Steiner tree can be decomposed, at those terminals with degree more than one, into smaller trees in which every terminal is a leaf. In such a decomposition, each resulting small tree is called a *full component*. The *size* of a full component is the number of terminals in it. A Steiner tree is *k-restricted* if every full component of it has a size at most k. The shortest k-restricted Steiner tree is also called the *k-restricted Steiner minimum tree*. Its length is denoted by $smt_k(P)$. Clearly, $smt_2(P)$ is the length of the minimum spanning tree on P, which is also denoted by $mst(P)$. Let $smt(P)$ denote the length of the Steiner minimum tree on P. If $smt_3(P)$ can be computed in polynomial-time, then it is better than $mst(P)$ for an approximation of $smt(P)$. However, so far no polynomial-time approximation has been found for $smt_3(P)$. Therefore, Zelikovsky [14] used a greedy approximation of $smt_3(P)$ to approximate $smt(P)$. Actually, Chang [4,5] used a similar greedy algorithm to compute an iterated 1-Steiner tree. Let $\mathcal{F}$ be a family of subgraphs of input edge-weighted graph G. For any connected subgraph H, denote by $mst(H)$ the length of the minimum spanning tree of H and for any subgraph H, denote by $mst(H)$ the sum of $mst(H')$ for H' over all connected components of H. Define

$$gain(H) = mst(P) - mst(P : H) - mst(H) ,$$

where $mst(P : H)$ is the length of the minimum spanning tree interconnecting all unconnected terminals in P after every edge of H shrinks into a point.

```
**Greedy Algorithm** $H \leftarrow \emptyset$;
  **while** $P$ has not been interconnected by $H$ **do**
    choose $F \in \mathcal{F}$ to maximize $gain(H \cup F)$;
  output $mst(H)$.
```

When $\mathcal{F}$ consists of all full components of size at most three, this greedy algorithm gives the 3-restricted Steiner tree of Zelikovsky [14]. When $\mathcal{F}$ consists of all 3-stars and all edges where a 3-star is a tree with three leaves and a central vertex, this greedy algorithm produces the iterated 1-Steiner tree. An interesting fact pointed out by Du et al. [9] is that the function $gain(\cdot)$ is submodular over all full components of size at most three, but not submodular over all 3-stars and edges.

Let us consider a base set E and a function f from all subsets of E to real numbers. f is *submodular* if for any two subsets A, B of E,

$$f(A) + f(B) \geq f(A \cup B) + f(A \cap B) .$$

For $x \in E$ and $A \subseteq E$, denote $\Delta_x f(A) = f(A \cup \{x\}) - f(A)$.

Lemma 1 *f is submodular if and only if for any $A \subset E$ and distinct $x, y \in E - A$,*

$$\Delta_x \Delta_y f(A) \leq 0 . \tag{1}$$

Proof Suppose f is submodular. Set $B = A \cup \{x\}$ and $C = A \cup \{y\}$. Then $B \cup C = A \cup A \cup \{x, y\}$ and $B \cap C = A$. Therefore, one has

$$f(A \cup \{x, y\}) - f(A \cup \{x\}) - f(A \cup \{y\}) + f(A) \leq 0 ,$$

that is, (1) holds.

Conversely, suppose (1) holds for any $A \subset E$ and distinct $x, y \in E - A$. Consider two subsets A, B of E. If $A \subseteq B$ or $B \subseteq A$, it is trivial to have

$$f(A) + f(B) \geq f(A \cup B) + f(A \cap B) .$$

Therefore, one may assume that $A \setminus B \neq \emptyset$ and $B \setminus A \neq \emptyset$. Write $A \setminus B = \{x_1, \dots, x_k\}$ and $B \setminus A = \{y_1, \dots, y_h\}$. Then

$$\begin{aligned} & f(A \cup B) - f(A) - f(B) + f(A \cap B) \\ &= \sum_{i=1}^{k} \sum_{j=1}^{h} \Delta_{x_i} \Delta_{y_j} f(A \cup \{x_1, \dots, x_{i-1}\} \cup \{y_1, \dots, y_{j-1}\}) \\ &\leq 0 , \end{aligned}$$

where $\{x_1, \dots x_{i-1}\} = \emptyset$ for $i = 1$ and $\{y_1, \dots, y_{j-1}\} = \emptyset$ for $j = 1$. □

Lemma 2 *Define $f(H) = -mst(P : H)$. Then f is submodular over edge set E.*

Proof Note that for any two distinct edges x and y not in subgraph H,

$$\begin{aligned} & \Delta_x \Delta f(H) \\ &= -mst(P : H \cup x \cup y) + mst(P : H \cup x) \\ &\quad + mst(P : H \cup y) - mst(P : H) \\ &= (mst(P : H) - mst(P : H \cup x \cup y)) \\ &\quad - (mst(P : H) - mst(P : H \cup x)) + (mst(P : H) \\ &\quad - mst(P : H \cup y)) . \end{aligned}$$

Let T be a minimum spanning tree for unconnected terminals after every edge of H shrinks into a point. T contains a path P_x connecting two endpoints of x and also a path P_y connecting two endpoints of y. Let e_x (e_y) be a longest edge in P_x (P_y). Then

$$\begin{aligned} & mst(P : H) - mst(P : H \cup x) = length(e_x) , \\ & mst(P : H) - mst(P : H \cup y) = length(e_y) . \end{aligned}$$

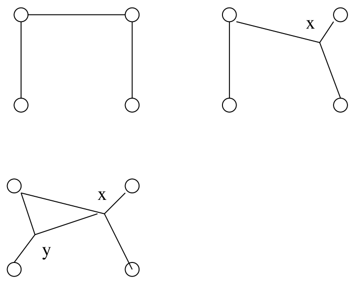
$mst(P : H) - mst(P : H \cup x \cup y)$ can be computed as follows: Choose a longest edge e' from $P_x \cup P_y$. Note that $T \cup x \cup y - e'$ contains a unique cycle Q. Choose a longest edge e'' from $(P_x \cup P_y) \cap Q$. Then

$$mst(P : H) - mst(P : H \cup x \cup y) = length(e') + length(e'').$$

Now, to show the submodularity of f, it suffices to prove

$$length(e_x) + length(e_y) \geq length(e') + length(e''). \quad (2)$$

Case 1. $e_x \notin P_x \cap P_y$ and $e_y \notin P_x \cap P_y$. Without loss of generality, assume $length(e_x) \geq length(e_y)$. Then one may choose $e' = e_x$ so that $(P_x \cup P_y) \cap Q = P_y$. Hence one can choose $e'' = e_y$. Therefore, the equality holds for (2).



Steiner Trees, Figure 2

Case 2. $e_x \notin P_x \cap P_y$ and $e_y \in P_x \cap P_y$. Clearly, $length(e_x) \geq length(e_y)$. Hence, one may choose $e' = e_x$ so that $(P_x \cup P_y) \cap Q = P_y$. Hence one can choose $e'' = e_y$. Therefore, the equality holds for (2).

Case 3. $e_x \in P_x \cap P_y$ and $e_y \notin P_x \cap P_y$. Similar to Case 2.

Case 4. $e_x \in P_x \cap P_y$ and $e_y \in P_x \cap P_y$. In this case, $length(e_x) = length(e_y) = length(e')$. Hence, (2) holds. □

The following explains that the submodularity of $gain(\cdot)$ holds for a k-restricted Steiner tree.

Proposition *Let $\mathcal{E}$ be the set of all full components of a Steiner tree. Then $gain(\cdot)$ as a function on the power set of $\mathcal{E}$ is submodular.*

Proof Note that for any $\mathcal{H} \subset \mathcal{E}$ and $x, y \in \mathcal{E} - \mathcal{H}$,

$$\Delta_x \Delta_y mst(H) = 0 ,$$

where $H = \cup_{z \in \mathcal{H}} z$. Thus, this proposition follows from Lemma 2. □

Let $\mathcal{F}$ be the set of 3-stars and edges chosen in the greedy algorithm to produce an iterated 1-Steiner tree. Then $gain(\cdot)$ may not be submodular on $\mathcal{F}$. To see this fact, consider two 3-stars x and y in Fig. 2. Note that $gain(x \cup y) > gain(x)$, $gain(y) \leq 0$ and $gain(\emptyset) = 0$. One has

$$gain(x \cup y) - gain(x) - gain(y) + gain(\emptyset) > 0 .$$

Applications

The Steiner tree problem is a classic NP-hard problem with many applications in the design of computer circuits, long-distance telephone lines, multicast routing in communication networks, etc. There exist many heuristics of the greedy-type for Steiner trees in the literature. Most of them have a good performance in computer experiments, without support from theoretical analysis. The approach given in this work may apply to them.

Open Problems

It is still open whether computing the 3-restricted Steiner minimum tree is NP-hard or not. For $k \geq 4$, it is known that computing the k-restricted Steiner minimum tree is NP-hard.

Cross References

► Greedy Approximation Algorithms
► Minimum Spanning Trees
► Rectilinear Steiner Tree

Recommended Reading

1. Arora, S.: Polynomial-time approximation schemes for Euclidean TSP and other geometric problems. Proc. 37th IEEE Symp. on Foundations of Computer Science, pp. 2–12 (1996)
2. Berman, P., Ramaiyer, V.: Improved approximations for the Steiner tree problem. J. Algorithms **17**, 381–408 (1994)
3. Bern, M., Plassmann, P.: The Steiner problem with edge lengths 1 and 2. Inf. Proc. Lett. **32**, 171–176 (1989)
4. Chang, S.K.: The generation of minimal trees with a Steiner topology. J. ACM **19**, 699–711 (1972)
5. Chang, S.K.: The design of network configurations with linear or piecewise linear cost functions. In: Symp. on Computer-Communications, Networks, and Teletraffic, pp. 363–369 IEEE Computer Society Press, California (1972)
6. Crourant, R., Robbins, H.: What Is Mathematics? Oxford University Press, New York (1941)
7. Du, D.Z., Hwang, F.K.: The Steiner ratio conjecture of Gilbert-Pollak is true. Proc. Natl. Acad. Sci. USA **87**, 9464–9466 (1990)
8. Du, D.Z., Zhang, Y., Feng, Q.: On better heuristic for euclidean Steiner minimum trees. In: Proceedings 32nd FOCS, IEEE Computer Society Press, California (1991)
9. Du, D.Z., Graham, R.L., Pardalos, P.M., Wan, P.J., Wu, W., Zhao, W.: Analysis of greedy approximations with nonsubmodular potential functions. In: Proceedings of 19th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 167–175. ACM, New York (2008)
10. Kahng, A., Robins, G.: A new family of Steiner tree heuristics with good performance: the iterated 1-Steiner approach. In: Proceedings of IEEE Int. Conf. on Computer-Aided Design, Santa Clara, pp.428–431 (1990)
11. Wolsey, L.A.: An analysis of the greedy algorithm for the submodular set covering problem. Combinatorica **2**, 385–393 (1982)
12. Robin, G., Zelikovsky, A.: Improved Steiner trees approximation in graphs. In: SIAM-ACM Symposium on Discrete Algorithms (SODA), San Francisco, CA, pp. 770–779. January (2000)
13. Smith, J.M., Lee, D.T., Liebman, J.S.: An $O(N \log N)$ heuristic for Steiner minimal tree problems in the Euclidean metric. Networks **11**, 23–39 (1981)
14. Zelikovsky, A.Z.: The 11/6-approximation algorithm for the Steiner problem on networks. Algorithmica **9**, 463–470 (1993)

Stochastic Scheduling

2001; Glazebrook, Nino-Mora

JAY SETHURAMAN
Industrial Engineering and Operations Research, Columbia University, New York, NY, USA

Keywords and Synonyms

Sequencing; Queueing

Problem Definition

Scheduling is concerned with the allocation of scarce resources (such as machines or servers) to competing activities (such as jobs or customers) over time. The distinguishing feature of a *stochastic* scheduling problem is that some of the relevant data are modeled as *random variables*, whose distributions are known, but whose actual realizations are not. Stochastic scheduling problems inherit several characteristics of their deterministic counterparts. In particular, there are virtually an unlimited number of problem types depending on the machine environment (single machine, parallel machines, job shops, flow shops), processing characteristics (preemptive versus non-preemptive; batch scheduling versus allowing jobs to arrive "over time"; due-dates; deadlines) and objectives (makespan, weighted completion time, weighted flow time, weighted tardiness). Furthermore, stochastic scheduling models have some new, interesting features (or difficulties!):

- The scheduler may be able to make inferences about the remaining processing time of a job by using information about its elapsed processing time; whether the scheduler is allowed to make use of this information or not is a question for the modeler.
- Many scheduling algorithms make decisions by comparing the processing times of jobs. If jobs have deterministic processing times, this poses no problems as there is only one way to compare them. If the processing times are random variables, comparing processing times is a subtle issue. There are many ways to compare pairs of random variables, and some are only *partial* orders. Thus any algorithm that operates by comparing processing times must now specify the particular ordering used to compare random variables (and to determine what to do if two random variables are not comparable under the specified ordering).

These considerations lead to the notion of a scheduling *policy*, which specifies how the scarce resources have to be allocated to the competing activities as a function of the *state* of the system at any point in time. The state of the system includes information such as prior job completions, the elapsed time of jobs currently in service, the realizations of the random release dates and due-dates (if any), and any other information that can be inferred based on the history observed so far. A policy that is allowed to make use of all this information is said to be *dynamic*, whereas a policy that is not allowed to use any state information is *static*.

Given any policy, the objective function for a stochastic scheduling model operating under that policy is typically a random variable. Thus comparison of two policies entails the comparison of the associated random variables, so the *sense* in which these random variables are compared must be specified. A common approach is to find a solution that optimizes the *expected value* of the objective function (which has the advantage that it is a total ordering); less commonly, other orderings such as the stochastic ordering or the likelihood ratio ordering are used.

Key Results

Consider a single machine that processes n jobs, with the (random) processing time of job i given by a distribution $F_i(\cdot)$ whose mean is p_i. The Weighted Shortest Expected Processing Time first (WSEPT) rule sequences the jobs in decreasing order of w_i/p_i. Smith [13] proved that the WSEPT rule minimizes the sum of weighted completion times when the processing times are deterministic. Rothkopf [11] generalized this result and proved the following:

Theorem 1 *The WSEPT rule minimizes the expected sum of the weighted completion times in the class of all nonpreemptive dynamic policies (and hence also in the class of all nonpreemptive static policies).*

If preemption is allowed, the WSEPT rule is not optimal. Nevertheless, Sevcik [12] showed how to assign an "index" to each job at each point in time such that scheduling a job with the largest index at each point in time is optimal. Such policies are called index policies and have been investigated extensively because they are (relatively) simple to implement and analyze. Often the optimality of index policies can be proved under some assumptions on the processing time distributions. For instance, Weber, Varaiya, and Walrand [14] proved the following result for scheduling n jobs on m identical parallel machines:

Theorem 2 *The SEPT rule minimizes the expected sum of completion times in the class of all nonpreemptive dynamic polices, if the processing time distributions of the jobs are stochastically ordered.*

For the same problem but with the makespan objective, Bruno, Downey, and Frederickson [3] proved the optimality of the Longest Expected Processing Time first rule provided all the jobs have exponentially distributed processing times.

One of the most significant achievements in stochastic scheduling is the proof of optimality of index policies for the multi-armed bandit problem and its many variants, due originally to Gittins and Jones [5,6]. In an instance of the bandit problem there are N projects, each of which is in any one of a possibly finite number of states. At each (discrete) time, any one of the projects can be attempted, resulting in a random reward; the attempted project undergoes a (Markovian) state-transition, whereas the other projects remain frozen and do not change state. The goal of the decision maker is to determine an optimal way to attempt the projects so as to maximize the total discounted reward. Of course one can solve this problem as a large, stochastic dynamic program, but such an approach does not reveal any structure, and is moreover computationally impractical except for very small problems. (Also, if the state space of any project is countable or infinite, it is not clear how one can solve the resulting DP exactly!) The remarkable result of Gittins and Jones [5] is the optimality of index policies: to each state of each project, one can associate an index so that attempting a project with the largest index at any point in time is optimal. The original proof of Gittins and Jones [5] has subsequently been simplified by many authors; moreover, several alternative proofs based on different techniques have appeared, leading to a much better understanding of the class of problems for which index policies are optimal.[2,4,6,10,17]

While index policies are easy to implement and analyze, they are often not optimal in many problems. It is therefore natural to investigate the gap between an optimal index policy (or a natural heuristic) and an optimal policy. For example, the WSEPT rule is a natural heuristic for the problem of scheduling jobs on identical parallel machines to minimize the expected sum of the weighted completion times. However, the WSEPT rule is not necessarily optimal. Weiss [16] showed that, under mild and reasonable assumptions, the expected number of times that the WSEPT rule differs from the optimal decision is bounded above by a *constant*, independent of the number of jobs. Thus, the WSEPT rule is asymptotically optimal. As another example of a similar result, Whittle [18] generalized the multi-armed bandit model to allow for state-transitions in projects that are *not* activated, giving rise to the "restless bandit" model. For this model, Whittle [18] proposed an index policy whose asymptotic optimality was established by Weber and Weiss [15].

A number of stochastic scheduling models allow for jobs to arrive over time according to a stochastic process. A commonly used model in this setting is that of a multiclass queueing network. Multiclass queueing networks serve as useful models for problems in which several types of *activities* compete for a limited number of shared *resources*. They generalize deterministic job-shop problems in two ways: jobs arrive *over time*, and each job has a *random* processing time at each stage. The optimal control problem in a multiclass queueing network is to find an optimal allocation of the available resources to activities over time. Not surprisingly, index policies are optimal only for restricted versions of this general model. An important example is scheduling a multiclass single-server system with feedback: there are N types of jobs, type i jobs arrive according to a Poisson process with rate λ_i, require service according to a service-time distribution $F_i(\cdot)$ with mean processing time s_i, and incur holding costs at rate c_i per unit time. A type i job after undergoing processing becomes a type j job with probability p_{ij}, or exits the system with probability $1 - \sum_j p_{ij}$. The objective is to find a scheduling policy that minimizes the expected holding cost rate in steady-state. Klimov [9] proved the optimality of index policies for this model, as well as for the objective in which the total discounted holding cost is to be minimized. While the optimality result does not hold when there are many parallel machines, Glazebrook and Niño-Mora [7] showed that this rule is asymptotically optimal. For more general models, the prevailing approach is to use approximations such as fluid approximations [1] or diffusion approximations [8].

Applications

Stochastic scheduling models are applicable in many settings, most prominently in computer and communication networks, call centers, logistics and transportation, and manufacturing systems [4,10].

Cross References

▶ List Scheduling
▶ Minimum Weighted Completion Time

Recommended Reading

1. Avram, F., Bertsimas, D., Ricard, M.: Fluid models of sequencing problems in open queueing networks: an optimal control approach. In: Kelly, F.P., Williams, R.J. (eds.) Stochastic Networks. Proceedings of the International Mathematics Association, vol. 71, pp. 199–234. Springer, New York (1995)
2. Bertsimas, D., Niño-Mora, J.: Conservation laws, extended polymatroids and multiarmed bandit problems: polyhedral approaches to indexable systems. Math. Oper. Res. **21**(2), 257–306 (1996)
3. Bruno, J., Downey, P., Frederickson, G.N.: Sequencing tasks with exponential service times to minimize the expected flow time or makespan. J. ACM **28**, 100–113 (1981)
4. Dacre, M., Glazebrook, K., Nino-Mora, J.: The achievable region approach to the optimal control of stochastic systems. J. R. Stat. Soc. Series B **61**(4), 747–791 (1999)
5. Gittins, J.C., Jones, D.M.: A dynamic allocation index for the sequential design experiments. In: Gani, J., Sarkadu, K., Vince, I. (eds.) Progress in Statistics. European Meeting of Statisticians I, pp. 241–266. North Holland, Amsterdam (1974)
6. Gittins, J.C.: Bandit processes and dynamic allocation indices. J. R. Stat. Soc. Series B, **41**(2), 148–177 (1979)
7. Glazebrook, K., Niño-Mora, J.: Parallel scheduling of multiclass M/M/m queues: approximate and heavy-traffic optimization of achievable performance. Oper. Res. **49**(4), 609–623 (2001)
8. Harrison, J.M.: Brownian models of queueing networks with heterogenous customer populations. In: Fleming, W., Lions, P.L. (eds.) Stochastic Differential Systems, Stochastic Control Theory and Applications. Proceedings of the International Mathematics Association, pp. 147–186. Springer, New York (1988)
9. Klimov, G.P.: Time-sharing service systems I. Theory Probab. Appl. **19**, 532–551 (1974)
10. Pinedo, M.: Scheduling: Theory, Algorithms and Systems, 2nd ed. Prentice Hall, Englewood Cliffs (2002)
11. Rothkopf, M.: Scheduling with Random Service Times. Manag. Sci. **12**, 707–713 (1966)
12. Sevcik, K.C.: Scheduling for minimum total loss using service time distributions. J. ACM **21**, 66–75 (1974)
13. Smith, W.E.: Various optimizers for single-stage production. Nav. Res. Logist. Quart. **3**, 59–66 (1956)
14. Weber, R.R., Varaiya, P., Walrand, J.: Scheduling jobs with stochastically ordered processing times on parallel machines to minimize expected flow time. J. Appl. Probab. **23**, 841–847 (1986)
15. Weber, R.R., Weiss, G.: On an index policy for restless bandits. J. Appl. Probab. **27**, 637–648 (1990)
16. Weiss, G.: Turnpike optimality of Smith's rule in parallel machine stochastic scheduling. Math. Oper. Res. **17**, 255–270 (1992)
17. Whittle, P.: Multiarmed bandit and the Gittins index. J. R. Stat. Soc. Series B **42**, 143–149 (1980)
18. Whittle, P.: Restless bandits: Activity allocation in a changing world. In: Gani, J. (ed.) A Celebration of Applied Probability. J Appl. Probab. **25A**, 287–298 (1988)

Strategyproof

▶ Nash Equilibria and Dominant Strategies in Routing
▶ Truthful Multicast

Stretch Factor

▶ Applications of Geometric Spanner Networks
▶ Dilation of Geometric Networks
▶ Geometric Dilation of Geometric Networks

String

▶ Compressed Pattern Matching
▶ Sequential Approximate String Matching
▶ Suffix Tree Construction in Hierarchical Memory
▶ Text Indexing

String Sorting

1997; Bentley, Sedgewick

Rolf Fagerberg
Department of Mathematics and Computer Science, University of Southern Denmark, Odense, Denmark

Keywords and Synonyms

Sorting of multi-dimensional keys; Vector sorting

Problem Definition

The problem is to sort a set of strings into lexicographical order. More formally: A *string* over an *alphabet* Σ is a finite sequence $x_1x_2x_3 \dots x_k$ where $x_i \in \Sigma$ for $i = 1, \dots, k$. The x_i's are called the *characters* of the string, and k is the *length* of the string. If the alphabet Σ is ordered, the *lexicographical order* on the set of strings over Σ is defined by declaring a string $x = x_1x_2x_3 \dots x_k$ smaller than a string $y = y_1y_2y_3 \dots y_l$ if either there exists a $j \geq 1$ such that $x_i = y_i$ for $1 \leq i < j$ and $x_j < y_j$, or if $k < l$ and $x_i = y_i$ for $1 \leq i \leq k$. Given a set S of strings over some ordered alphabet, the problem is to sort S according to lexicographical order.

The input to the string sorting problem consists of an array of pointers to the strings to be sorted. The output is a permutation of the array of pointers, such that traversing the array will point to the strings in non-decreasing lexicographical order.

The complexity of string sorting depends on the alphabet as well as the machine model. The main solution [15] described in this entry works for alphabets of unbounded size (i. e., comparisons are the only operations on characters of Σ), and can be implemented on a pointer machine. See below for more information on the asymptotic complexity of string sorting in various settings.

Key Results

This section is structured as follows: first the key result appearing in title of this entry [15] is described, then an overview of other relevant results in the area of string sorting is given.

The string sorting algorithm proposed by Bentley and Sedgewick in 1997 [15] is called Three-Way Radix Quicksort [5]. It works for unbounded alphabets, for which it achieves optimal performance.

Theorem 1 *The algorithm Three-Way Radix Quicksort sorts K strings of total length N in time $O(K \log K + N)$.*

That this time complexity is optimal follows by considering strings of the form `bbb...bx`, where all x's are different: Sorting the strings can be no faster than sorting the x's, and all `b`'s must be read (else an adversary could change one unread `b` to `a` or `c`, making the returned order incorrect). A more precise version of the bounds above (upper as well as lower) is $K \log K + D$, where D is the sum of the lengths of the *distinguishing prefixes* of the strings. The distinguishing prefix d_s of a string s in a set S is the shortest prefix of s which is not a prefix of another string in S (or is s itself, if s is a prefix of another string). Clearly, $K \leq D \leq N$.

The Three-Way Radix Quicksort of Bentley and Sedgewick is not the first algorithm to achieve this complexity—however, it is a very simple and elegant way of doing it. As demonstrated in [3,15], it is also very fast in practice. Although various elements of the algorithm had been noted earlier, their practical usefulness for string sorting was overlooked until the work in [15].

Three-Way Radix Quicksort is shown in pseudo-code in Fig. 1 (adapted from [5]), where S is a list of strings to be sorted and d is an integer. To sort S, an initial call Sort$(S, 1)$ is made. The value s_d denotes the dth character of the string s, and $+$ denotes concatenation. The presentation in Fig. 1 assumes that all strings end in a special

```
Sort$(S, d)$
    IF $|S| \leq 1$:
        RETURN
    Choose a partitioning character
    $v \in \{s_d \mid s \in S\}$
    $S_< = \{s \in S \mid s_d < v\}$
    $S_= = \{s \in S \mid s_d = v\}$
    $S_> = \{s \in S \mid s_d > v\}$
    Sort$(S_<, d)$
    IF $v \neq$ EOS:
        Sort$(S_=, d + 1)$
    Sort$(S_>, d)$
    $S = S_< + S_= + S_>$
```

String Sorting, Figure 1
Three-Way Radix Quicksort (assuming each string ends in a special EOS character)

End-Of-String (EOS) character (such as the null character in C). In an actual implementation, S will be an array of pointers to strings, and the sort will in-place (using an in-place method from standard Quicksort for three-way partitioning of the array into segments holding $S_<$, $S_=$, and $S_>$), rendering concatenation superfluous.

Correctness follows from the following invariant being maintained by the algorithm: At the start of a call $\textsc{Sort}(S, d)$, all strings in S agree on the first $d - 1$ characters.

Time complexity depends on how the partitioning character v is chosen. One particular choice is the median of all the dth characters (including doublets) of the strings in S. Partitioning and median finding can be done in time $O(|S|)$, which is $O(1)$ time per string partitioned. Hence, the total running time of the algorithm is the sum over all strings of the number of partitionings they take part in. For each string, let a partitioning be of type I if the string ends up in $S_<$ or $S_>$, and of type II if it ends up in $S_=$. For a string s, type II can only occur $|d_s|$ times and type I can only occur $\log K$ times. Hence, the running time is $O(K \log K + D)$.

Like for standard Quicksort, median finding impairs the constant factors of the algorithm, and more practical choices of partitioning character include selecting a random element among all the dth characters of the strings in S, and selecting the median of three elements in this set. The worst-case bound is lost, but the result is a fast, randomized algorithm.

Note that the ternary recursion tree of Three-Way Radix Quicksort is equivalent to a trie over the strings sorted, with trie nodes implemented by binary trees (where the elements stored in a binary tree are the characters of the trie edges leaving the trie node). The equivalence is as follows: an edge representing a recursive call on $S_<$ or $S_>$ corresponds to an edge of a binary tree (implementing a trie node), and an edge representing a recursive call on $S_=$ corresponds to a trie edge leading to a child node in the trie. This trie implementation is named Ternary Search Trees in [15]. Hence, Three-Way Radix Quicksort may additionally be viewed as a construction algorithm for an efficient dictionary structure for strings.

For the version of the algorithm where the partitioning character v is chosen as the median of all the dth characters, it is not hard to see that the binary trees representing the trie nodes become weighted trees, i. e., binary trees in which each element x has an associated weight w_x, and searches for x takes $O(\log W/w_x)$, where $W = \Sigma_x w_x$ is the sum of all weights in the binary tree. The weight of a binary tree node storing character x is the number of strings in the trie which reside below the trie edge labeled with character x and leaving the trie node represented by the binary tree. As shown in [13], in such a trie implementation searching for a string P among K stored strings takes time $O(\log K + |P|)$, which is optimal for unbounded (i. e., comparison-based) alphabets.

Other key results in the area of string sorting are now described. The classic string sorting algorithm is Radixsort, which assumes a constant sized alphabet. The Least-Significant-Digit-first variant is easy to implement, and runs in $O(N + l|\Sigma|)$ time, where l is the length of the longest string. The Most-Significant-Digit-first variant is more complicated to implement, but has a better running time of $O(D + d|\Sigma|)$, where D is the sum of the lengths of the distinguishing prefixes, and d is the longest distinguishing prefix. [12] discusses in depth efficient implementations of Radixsort.

If the alphabet consists of integers, then on a word-RAM the complexity of string sorting is essentially determined by the complexity of integer sorting. More precisely, the time (when allowing randomization) for sorting strings is $\Theta(\text{Sort}_{\text{Int}}(K) + N)$, where $\text{Sort}_{\text{Int}}(K)$ is the time to sort K integers [2], which currently is known to be $O(K\sqrt{\log\log K})$ [11].

Returning to comparison-based model, the papers [8,10] give generic methods for turning any data structure over one-dimensional keys into a data structure over strings. Using finger search trees, this gives an adaptive sorting method for strings which uses $O(N + K \log(F/K))$ time, where F is the number of inversions among the strings to be sorted.

Concerning space complexity, it has been shown [9] that string sorting can still be done in $O(K \log K + N)$ time using only $O(1)$ space besides the strings themselves. However, this assumes that all strings have equal lengths.

All algorithms so far are designed to work in internal memory, where CPU time is assumed to be the dominating factor. For external memory computation, a more relevant cost measure is the number of I/Os performed, as captured by the I/O-model [1], which models a two-level memory hierarchy with an infinite outer memory, an inner memory of size M, and transfer (I/Os) between the two levels taking place in blocks of size B. In external memory, upper bounds were first given in [4], along with matching lower bounds in restricted I/O-models. For a comparison based model where strings may only be moved in blocks of size B (hence, characters may not be moved individually), it is shown that string sorting takes $\Theta(N_1/B \log_{M/B}(N_1/B) + K_2 \log_{M/B} K_2 + N/B)$ I/Os, where N_1 is the total length of strings shorter than B characters, K_2 is the number of strings of at least B characters, and N is the total number of characters. This bound is equal to the sum of the I/O costs of sorting the characters of the short strings, sorting B characters from each of the long strings, and scanning all strings. In the same paper, slightly better bounds in a model where characters may be moved individually in internal memory are given, as well as some upper bounds for non-comparison based string sorting. Further bounds (using randomization) for non-comparison based string sorting have been given, with I/O bounds of $O(K/B \log_{M/B}(K/M) \log\log_{M/B}(K/M) + N/B)$ [7] and[1] $O(K/B(\log_{M/B}(N/M))^2 \log_2 K + N/B)$.

Returning to internal memory, it may also there be the case that memory hierarchy effects are the determining factor for the running time of algorithms, but now due to cache faults rather than disk I/Os. Heuristic algorithms (i. e., algorithms without good worst case bounds), aiming at minimizing cache faults for internal memory string sorting, have been developed. Of these, the Burstsort line of algorithms [16] have particularly promising experimental results reported.

Applications

Data sets consisting partly or entirely of string data are very common: Most database applications have strings as one of the data types used, and in some areas, such as bioinformatics, web retrieval, and word processing, string data is predominant. Additionally, strings form a general and fundamental data model, containing e. g. integers and multi-dimensional data as special cases. Since sorting is arguably among the most important data processing tasks in any domain, string sorting is a general and important problem with wide practical applications.

Open Problems

As appears from the bounds discussed above, the asymptotic complexity of the string sorting problem is known for comparison based alphabets. For integer alphabets on the word-RAM, the problem is almost closed in the sense that it is equivalent to integer sorting, for which the gap left between the known bounds and the trivial linear lower bound is small.

In external memory, the situation is less settled. As noted in [4], a natural upper bound to hope for in a comparison based setting is to meet the lower bound of $\Theta(K/B \log_{M/B} K/M + N/B)$ I/Os, which is the sorting bound for K single characters plus the complexity of scanning the input. The currently known upper bounds only gets close to this if leaving the comparison based setting and allowing randomization.

Further open problems include adaptive sorting algorithms for other measures of presortedness than that used in [8,10], and algorithms for sorting general strings (not necessarily of equal lengths) using only $O(1)$ additional space [9].

Experimental Results

In [15], experimental comparison of two implementations (one simple and one tuned) of Three-Way Radix Quicksort with a tuned Quicksort [6] and a tuned Radixsort [12] showed the simple implementation to always outperform the Quicksort implementation, and the tuned implementation to be competitive with the Radixsort implementation.

In [3], experimental comparison among existing and new Radixsort implementations (including the one used in [15]), as well as tuned Quicksort and tuned Three-Way Radix Quicksort was performed. This study confirms the picture of Three-Way Radix Quicksort as very competitive, always being one of the fastest algorithms, and arguably the most robust across various input distributions.

Data Sets

The data sets used in [15]: http://www.cs.princeton.edu/~rs/strings/. The data sets used in [3]: http://www.jea.acm.org/1998/AnderssonRadixsort/.

URL to Code

Code in C from [15]:
http://www.cs.princeton.edu/~rs/strings/.
Code in C from [3]:
http://www.jea.acm.org/1998/AnderssonRadixsort/.
Code in Java from [14]:
http://www.cs.princeton.edu/~rs/Algs3.java1-4/code.txt.

Cross References

- ► Suffix Array Construction
- ► Suffix Tree Construction in Hierarchical Memory
- ► Suffix Tree Construction in RAM

Recommended Reading

1. Aggarwal, A., Vitter, J.S.: The input/output complexity of sorting and related problems. Commun. ACM **31**, 1116–1127 (1988)
2. Andersson, A., Nilsson, S.: A new efficient radix sort. In: Proceedings of the 35th Annual Symposium on Foundations of

[1] Ferragina, personal communication.

Computer Science (FOCS '94), IEEE Comput. Soc. Press, pp. 714–721 (1994)

3. Andersson, A., Nilsson, S.: Implementing radixsort. ACM J. Exp. Algorithmics **3**, 7 (1998)
4. Arge, L., Ferragina, P., Grossi, R., Vitter, J.S.: On sorting strings in external memory (extended abstract). In: Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC '97), ACM, ed., pp. 540–548. ACM Press, El Paso (1997),
5. Bentley, J., Sedgewick, R.: Algorithm alley: Sorting strings with three-way radix quicksort. Dr. Dobb's J. Softw. Tools **23**, 133–134, 136–138 (1998)
6. Bentley, J.L., McIlroy, M.D.: Engineering a sort function. Softw. Pract. Exp. **23**, 1249–1265 (1993)
7. Fagerberg, R., Pagh, A., Pagh, R.: External string sorting: Faster and cache-oblivious. In: Proceedings of STACS '06. LNCS, vol. 3884, pp. 68–79. Springer, Marseille (2006)
8. Franceschini, G., Grossi, R.: A general technique for managing strings in comparison-driven data structures. In: Proceedings of the 31st International Colloquium on Automata, Languages and Programming (ICALP '04). LNCS, vol. 3142, pp. 606–617. Springer, Turku (2004)
9. Franceschini, G., Grossi, R.: Optimal in-place sorting of vectors and records. In: Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP '05). LNCS, vol. 3580, pp. 90–102. Springer, Lisbon (2005)
10. Grossi, R., Italiano, G.F.: Efficient techniques for maintaining multidimensional keys in linked data structures. In: Proceedings of the 26th International Colloquium on Automata, Languages and Programming (ICALP '99). LNCS, vol. 1644, pp. 372–381. Springer, Prague (1999)
11. Han, Y., Thorup, M.: Integer sorting in $O(n\sqrt{\log\log n})$ expected time and linear space. In: Proceedings of the 43rd Annual Symposium on Foundations of Computer Science (FOCS '02), pp. 135–144. IEEE Computer Society Press, Vancouver (2002)
12. McIlroy, P.M., Bostic, K., McIlroy, M.D.: Engineering radix sort. Comput. Syst. **6**, 5–27 (1993)
13. Mehlhorn, K.: Dynamic binary search. SIAM J. Comput. **8**, 175–198 (1979)
14. Sedgewick, R.: Algorithms in Java, Parts 1–4, 3rd edn. Addison-Wesley, (2003)
15. Sedgewick, R., Bentley, J.: Fast algorithms for sorting and searching strings. In: Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '97), ACM, ed., pp. 360–369. ACM Press, New Orleans (1997)
16. Sinha, R., Zobel, J., Ring, D.: Cache-efficient string sorting using copying. ACM J. Exp. Algorithmics. **11** (2006)

Substring Parsimony

2001; Blanchette, Schwikowski, Tompa

MATHIEU BLANCHETTE
Department of Computer Science, McGill University, Montreal, QC, Canada

Problem Definition

The Substring Parsimony Problem, introduced by Blanchette et al. [1] in the context of motif discovery in biological sequences, can be described in a more general framework:

Input:

- A discrete space S on which an integral distance d is defined (i. e. $d(x, y) \in \mathbb{N}\ \forall x, y \in S$).
- A rooted binary tree $T = (V, E)$ with n leaves. Vertices are labeled $\{1, 2, \ldots, n, \ldots, |V|\}$, where the leaves are vertices $\{1, 2, \ldots, n\}$.
- Finite sets $S_1, S_2, \ldots, S_n$, where set $S_i \subseteq S$ is assigned to leaf i, for all $i = 1 \ldots n$.
- A non-negative integer t

Output: All solutions of the form $(x_1, x_2, \ldots, x_n, \ldots, x_{|V|})$ such that:

- $x_i \in S$ for all $i = 1 \ldots |V|$
- $x_i \in S_i$ for all $i = 1 \ldots n$
- $\sum_{(u,v)\in E} d(x_u, x_v) \le t$

The problem thus consists of choosing one element x_i from each set S_i such that the Steiner distance of the set of points is at most t. This is done on a Steiner tree T of fixed topology. The case where $|S_i| = 1$ for all $i = 1 \ldots n$ is a standard Steiner tree problem on a fixed tree topology (see [11]). It is known as the Maximum Parsimony Problem and its complexity depends on the space S.

Key Results

The substring parsimony problem can be solved using a dynamic programming algorithm. Let $u \in V$ and $s \in S$. Let $W_u[s]$ be the score of the best solution that can be obtained for the subtree rooted at node u, under the constraint that node u is labeled with s, i. e.

$$W_u[s] = \min_{\substack{x_1,\ldots,x_{|V|}\in S\\ x_u=s}} \sum_{\substack{(i,j)\in E\\ i,j\in \text{subtree}(u)}} d(x_i, x_j)\,.$$

Let v be a child of u, and let $X_{(u,v)}[s]$ be the score of the best solution that can be obtained for the subtree consisting of node u together with the subtree rooted at its child v, under the constraint that node u is labeled with s:

$$X_{(u,v)}[s] = \min_{\substack{x_1,\ldots,x_{|V|}\in S\\ x_u=s}} \sum_{\substack{(i,j)\in E\\ i,j\in \text{subtree}(v)\cup\{(u,v)\}}} d(x_i, x_j)\,.$$

Then, we have:

$$W_u[s] = \begin{cases} 0 & \text{if } u \text{ is a leaf and } s \in S_u \\ +\infty & \text{if } u \text{ is a leaf and } s \notin S_u \\ \sum_{v\in\text{children}(u)} X_{(u,v)}[s] & \text{if } u \text{ is not a leaf} \end{cases}$$

and

$$X_{(u,v)}[s] = \min_{y'\in S} W_u[s'] + d(s, s')\,.$$

Tables W and X can thus be computed using a dynamic programming algorithm, proceeding in a post-order traversal of the tree. Solutions can then be recovered by tracing the computation back for all s such that $W_{\text{root}}[s] \leq t$. Note that the same solution may be recovered more than once in this process.

A straight-forward implementation of this dynamic programming algorithm would run in time $O(n \cdot |S|^2 \cdot \gamma(S))$, where $\gamma(S)$ is the time needed to compute the distance between any two points in S. Let $N_a(S)$ be the maximum number of a-neighbors a point in S can have, i. e. $N_a(S) = \max_{x \in S} |\{y \in S : d(x, y) = a\}|$. Blanchette et al. [3] showed how to use a modified breadth-first search of the space S to compute each table $X_{(u,v)}$ in time $O(|S| \cdot N_1(S))$, thus reducing the total time complexity to $O(n \cdot |S| \cdot N_1(S))$. Since only solutions with a score of at most t are of interest, the complexity can be further reduced by only computing those table entries which will yield a score of at most t. This results in an algorithm whose running time is $O(n \cdot M \cdot N_{\lfloor t/2 \rfloor}(S) \cdot N_1(S))$ where $M = \max_{i=1\dots n} |S_i|$.

The problem has been mostly studied in the context of biological sequence analysis, where $S = \{A, C, G, T\}^k$, for some small k ($k = 5, \dots, 20$ are typical values). The distance d is the Hamming distance, and a phylogenetic tree T is given. The case where $|S_i| = 1$ for all $i = 1 \dots n$ is known as the Maximum Parsimony Problem and can be solved in time $O(n \cdot k)$ using Fitch's algorithm [9] or Sankoff's algorithm [12]. In the more general version, a long DNA sequence P_u of length L is assigned to each leaf u. The set S_u is defined as the set of all k-substrings of P_u. In this case, $M = L - k + 1 \in O(L)$, and $N_a \in O(\min(4^k, (3k)^a))$, resulting in a complexity of $O(n \cdot L \cdot 3k \cdot \min(4^k, (3k)^{\lfloor d/2 \rfloor}))$. Notice that for a fixed k and d, the algorithm is linear over the whole sequence. The problem was independently shown to be NP-hard by Blanchette et al. [3] and by Elias [7].

Applications

Most applications are found in computational biology, although the algorithm can be applied to a wide variety of domains. The algorithm for the substring parsimony problem has been implemented in a software package called FootPrinter [5] and applied to the detection of transcription factor binding sites in orthologous DNA regulatory sequences through a method called phylogenetic footprinting [4]. Other applications include the search for conserved RNA secondary structure motifs in orthologous RNA sequences [2]. Variants of the problem have been defined to identify motifs regulating alternative splicing [13]. Blanchette et al. [3] study a relaxation of the problem where one does not require that a substring be chosen from each of the input sequences, but instead asks that substrings be chosen from a sufficiently large subset of the input sequence. Fang and Blanchette [8] formulate another variant of the problem where substring choices are constrained to respect a partial order relation defined by a set of local multiple sequence alignments.

Open Problems

Optimizations taking advantage of the specific structure of the space S may yield more efficient algorithms in certain cases. Many important variations could be considered. First, the case where the tree topology is not given needs to be considered, although the resulting problems would usually be NP-hard even when $|S_i| = 1$. Another important variation is one where the phylogenetic relationships between trees is not given by a tree but rather by a phylogenetic network [10]. Finally, randomized algorithms similar to those proposed by Buhler et al. [6] may yield important and practical improvements.

URL to Code

http://bio.cs.washington.edu/software.html

Cross References

- ▶ Closest Substring
- ▶ Efficient Methods for Multiple Sequence Alignment with Guaranteed Error Bounds
- ▶ Local Alignment (with Affine Gap Weights)
- ▶ Local Alignment (with Concave Gap Weights)
- ▶ Statistical Multiple Alignment
- ▶ Steiner Trees

Recommended Reading

1. Blanchette, M.: Algorithms for phylogenetic footprinting. In: RECOMB01: Proceedings of the Fifth Annual International Conference on Computational Molecular Biology, pp. 49–58. ACM Press, Montreal (2001)
2. Blanchette, M.: Algorithms for phylogenetic footprinting. Ph. D. thesis, University of Washington (2002)
3. Blanchette, M., Schwikowski, B., Tompa, M.: Algorithms for phylogenetic footprinting. J. Comput. Biol. **9**(2), 211–223 (2002)
4. Blanchette, M., Tompa, M.: Discovery of regulatory elements by a computational method for phylogenetic footprinting. Genome Res. **12**, 739–748 (2002)
5. Blanchette, M., Tompa, M.: Footprinter: A program designed for phylogenetic footprinting. Nucleic Acids Res. **31**(13), 3840–3842 (2003)
6. Buhler, J., Tompa, M.: Finding motifs using random projections. In: RECOMB01: Proceedings of the Fifth Annual Interna-

tional Conference on Computational Molecular Biology, 2001, pp. 69–76
7. Elias, I.: Settling the intractability of multiple alignment. J. Comput. Biol. **13**, 1323–1339 (2006)
8. Fang, F., Blanchette, M.: Footprinter3: phylogenetic footprinting in partially alignable sequences. Nucleic Acids Res. **34**(2), 617–620 (2006)
9. Fitch, W.M.: Toward defining the course of evolution: Minimum change for a specified tree topology. Syst. Zool. **20**, 406–416 (1971)
10. Huson, D.H., Bryant, D.: Application of phylogenetic networks in evolutionary studies. Mol. Biol. Evol. **23**(2), 254–267 (2006)
11. Sankoff, D., Rousseau, P.: Locating the vertices of a Steiner tree in arbitrary metric space. Math. Program. **9**, 240–246 (1975)
12. Sankoff, D.D.: Minimal mutation trees of sequences. SIAM J. Appl. Math. **28**, 35–42 (1975)
13. Shigemizu, D., Maruyama, O.: Searching for regulatory elements of alternative splicing events using phylogenetic footprinting. In: Proceedings of the Fourth Workshop on Algorithms for Bioinformatics. Lecture Notes in Computer Science, pp. 147–158. Springer, Berlin (2004)

Succinct Data Structures for Parentheses Matching

2001; Munro, Raman

MENG HE
School of Computer Science, University of Waterloo, Waterloo, ON, Canada

Keywords and Synonyms

Succinct balanced parentheses

Problem Definition

This problem is to design succinct representation of balanced parentheses in a manner in which a number of "natural" queries can be supported quickly, and use it to represent trees and graphs succinctly. The problem of succinctly representing balanced parentheses was initially proposed by Jacobson [6] in 1989, when he proposed *succinct data structures*, i. e. data structures that occupy space close to the information-theoretic lower bound to represent them, while supporting efficient navigational operations. Succinct data structures provide solutions to manipulate large data in modern applications. The work of Munro and Raman [8] provides an optimal solution to the problem of balanced parentheses representation under the word RAM model, based on which they design succinct trees and graphs.

Balanced Parentheses

Given a balanced parenthesis sequence of length $2n$, where there are n opening parentheses and n closing parentheses, consider the following operations:

- `findclose`(i) (`findopen`(i)), the matching closing (opening) parenthesis for the opening (closing) parenthesis at position i;
- `excess`(i), the number of opening parentheses minus the number of closing parentheses in the sequence up to (and including) position i;
- `enclose`(i), the closest enclosing (matching parenthesis) pair of a given matching parenthesis pair whose opening parenthesis is at position i.

Trees

There are essentially two forms of trees. An *ordinal tree* is a rooted tree in which the children of a node are ordered and specified by their ranks, while in a *cardinal tree* of degree k, each child of a node is identified by a unique number from the set $\{1, 2, \cdots, k\}$. An *binary tree* is a cardinal tree of degree 2. The information-theoretic lower bound of representing an ordinal tree or binary tree of n nodes is $2n - o(n)$ bits, as there are $\binom{2n}{n}/(n+1)$ different ordinal trees or binary trees.

Consider the following operations on ordinal trees (a node is referred to by its preorder number):

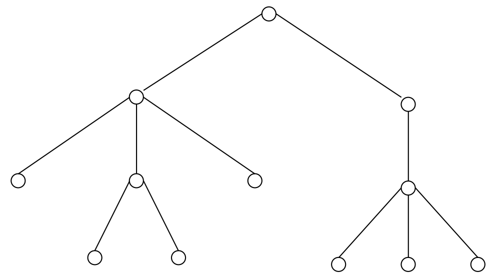
- `child`(x,i), the ith child of node x for $i \geq 1$;
- `child_rank`(x), the number of left siblings of node x;
- `depth`(x), the depth of x, i. e. the number of edges in the rooted path to node x;
- `parent`(x), the parent of node x;
- `nbdesc`(x), the number of descendants of node x;
- `height`(x), the height of the subtree rooted at node x;
- `LCA`(x,y), the lowest common ancestor of node x and node y.

On binary trees, the operations `parent`, `nbdesc` and the following operations are considered:

- `leftchild`(x) (`rightchild`(x)), the left (right) child of node x.

Graphs

Consider an undirected graph G of n vertices and m edges. Bernhart and Kainen [1] introduced the concept of *page book embedding*. A *k-book embedding* of a graph is a topological embedding of it in a book of k pages that specifies the ordering of the vertices along the spine, and carries each edge into the interior of one page, such that the edges on a given page do not intersect. Thus, a graph with



Balanced parentheses: ((( ) (( ) ( )) ( )) ((( ) ( ) ( ))))

Succinct Data Structures for Parentheses Matching, Figure 1
An example of the balanced parenthesis sequence of a given ordinal tree

one page is an *outerplanar graph*. The *pagenumber* or *book thickness* [1] of a graph is the minimum number of pages that the graph can be embedded in. A very common type of graphs are planar graphs, and any planar graph can be embedded in at most 4 pages [15]. Consider the following operations on graphs:

- adjacency(x,y), whether vertices x and y are adjacent;
- degree(x), the degree of vertex x;
- neighbors(x), the neighbors of vertex x.

Key Results

All the results cited are under the word RAM model with word size $\Theta(\lg n)$ bits[1], where n is the size of the problem considered.

Theorem 1 ([8]) *A sequence of balanced parentheses of length $2n$ can be represented using $2n + o(n)$ bits to support the operations findclose, findopen, excess and enclose in constant time.*

There is a polymorphism between a balanced parenthesis sequence and an ordinal tree: when performing a depth-first traversal of the tree, output an opening parenthesis each time a node is visited, and a closing parenthesis immediately after all the descendants of a node are visited (see Fig. 1 for an example). The work of Munro and Raman proposes a succinct representation of ordinal trees using $2n + o(n)$ bits to support depth, parent and nbdesc in constant time, and child(x,i) in $O(i)$ time. Lu and Yeh have further extended this representation to support child, child_rank, height and LCA in constant time.

[1] lg n denotes $\lceil \log_2 n \rceil$.

Theorem 2 ([8,7]) *An ordinal tree of n nodes can be represented using $2n + o(n)$ bits to support the operations child, child_rank, parent, depth, nbdesc, height and LCA in constant time.*

A similar approach can be used to represent binary trees:

Theorem 3 ([8]) *A binary tree of n nodes can be represented using $2n + o(n)$ bits to support the operations leftchild, rightchild, parent and nbdesc in constant time.*

Finally, balanced parentheses can be used to represent graphs. To represent a one-page graph, the work of Munro and Raman proposes to list the vertices from left to right along the spine, and each node is represented by a pair of parentheses, followed by zero or more closing parentheses and then zero or more opening parentheses, where the number of closing (or opening) parentheses is equal to the number of adjacent vertices to its left (or right) along the spine (see Fig. 2 for an example). This representation can be applied to each page to represent a graph with pagenumber k.

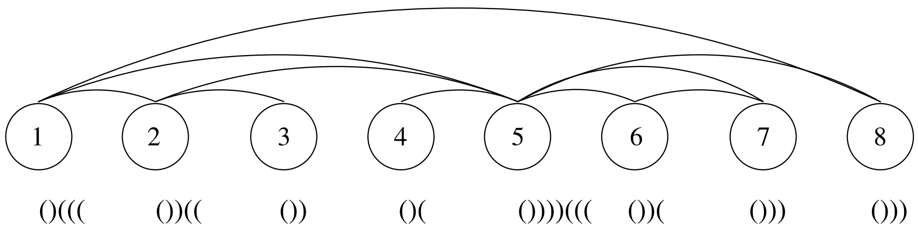
Theorem 4 ([8]) *An outerplanar graph of n vertices and m edges can be represented using $2n + 2m + o(n + m)$ bits to support operations adjacency and degree in constant time, and neighbors(x) in time proportional to the degree of x.*

Theorem 5 ([8]) *A graph of n vertices and m edges with pagenumber k can be represented using $2kn+2m+o(nk+m)$ bits to support operations adjacency and degree in $O(k)$ time, and neighbors(x) in $O(d(x) + k)$ time where $d(x)$ is the degree of x. In particular, a planar graph of n vertices and m nodes can be represented using $8n + 2m + o(n)$ bits to support operations adjacency and degree in constant time, and neighbors(x) in $O(d(x))$ time where $d(x)$ is the degree of x.*

Applications

Succinct Representation of Suffix Trees

As a result of the growth of the textual data in databases and on the World Wide Web, and also applications in bioinformatics, various indexing techniques have been developed to facilitate pattern searching. Suffix trees [14] are a popular type of text indexes. A suffix tree is constructed over the suffixes of the text as a tree-based data structure, so that queries can be performed by searching the suffixes of the text. It takes $O(m)$ time to use a suffix tree to check whether an arbitrary pattern P of length m is a substring of a given text T of length n, and to count the number of the occurrences, occ, of P in T. $O(occ)$ additional time



Succinct Data Structures for Parentheses Matching, Figure 2
An example of the balanced parenthesis sequence of a graph with one page

is required to list all the occurrences of P in T. However, a standard representation of a suffix tree requires somewhere between $4n \lg n$ and $6n \lg n$ bits, which is impractical for many applications.

By reducing the space cost of representing the tree structure of a suffix tree (using the work of Munro and Raman), Munro, Raman and Rao [9] have designed space-efficient suffix trees. Given a string of n characters over a fixed alphabet, they can represent a suffix tree using $n \lg n + O(n)$ bits to support the search of a pattern in $O(m + occ)$ time. To achieve this result, they have also extended the work of Munro and Raman to support various operations to retrieve the leaves of a given subtree in an ordinal tree. Based on similar ideas and by applying compressed suffix arrays [5], Sadakane [13] has proposed a different trade-off; his compressed suffix tree occupies $O(n \lg \sigma)$ bits, where σ is the size of the alphabet, and can support any algorithm on a suffix tree with a slight slowdown of a factor of polylog(n).

Succinct Representation of Functions

Munro and Rao [11] have considered the problem of succinctly representing a given function, $f : [n] \to [n]$, to support the computation of $f^k(i)$ for an arbitrary integer k. The straightforward representation of a function is to store the sequence $f(i)$, for $i = 0, 1, \ldots, n-1$. This takes $n \lg n$ bits, which is optimal. However, the computation of $f^k(i)$ takes $\Theta(k)$ time even in the easier case when k is positive. To address this problem, Munro and Rao [11] first extends the representation of balanced parenthesis to support the `next_excess`(i,k) operator, which returns the minimum j such that $j > i$ and `excess`(j) = k. They further use this operator to support the `level_anc`(x,i) operator on succinct ordinal trees, which returns the ith ancestor of node x for $i \geq 0$ (given a node x at depth d, its ith ancestor is the ancestor of x at depth $d - i$). Then, using succinct ordinal trees with the support for `level_anc`, they propose a succinct representation of functions using $(1 + \epsilon)n \lg n + O(1)$ bits for any fixed positive constant ϵ, to support $f^k(i)$ in constant time when $k > 0$, and $f^k(i)$ in $O(1 + |f^k(i)|)$ time when $k < 0$.

Multiple Parentheses and Graphs

Chuang et al. [3] have proposed to succinctly represent *multiple parentheses*, which is a string of $O(1)$ types of parentheses that may be unbalanced. They have extended the operations on balanced parentheses to multiple parentheses and designed a succinct representation. Based on the properties of canonical orderings for planar graphs, they have used multiple parentheses and the succinct ordinal trees to represent planar graphs. One of their main results is a succinct representation of planar graphs of n vertices and m edges in $2m + (5 + \epsilon)n + o(m + n)$ bits, for any constant $\epsilon > 0$, to support the operations supported on planar graphs in Theorem 5 in asymptotically the same amount of time. Chiang et al. [2] have further reduced the space cost to $2m + 3n + o(m + n)$ bits. In their paper, they have also shown how to support the operation `wrapped`(i), which returns the number of matching parenthesis pairs whose closest enclosing (matching parenthesis) pair is the pair whose opening parenthesis is at position i, in constant time on balanced parentheses. They have used it to show how to support the operation `degree`(x), which returns the degree of node x (i. e. the number of its children), in constant time on succinct ordinal trees.

Open Problems

One open research area is to support more operations on succinct trees. For example, it is not known how to support the operation to convert a given node's rank in a preorder traversal into its rank in a level-order traversal.

Another open research area is to further reduce the space cost of succinct planar graphs. It is not known whether it is possible to further improve the encoding of Chiang et al. [2].

A third direction for future work is to design succinct representations of dynamic trees and graphs. There have been some preliminary results by Munro et al. [10] on succinctly representing dynamic binary trees, which have been further improved by Raman and Rao [12]. It may be possible to further improve these results, and there are other related dynamic data structures that do not have succinct representations.

Experimental Results

Geary et al. [4] have engineered the implementation of succinct ordinal trees based on balanced parentheses. They have performed experiments on large XML trees. Their implementation uses orders of magnitude less space than the standard pointed-based representation, while supporting tree traversal operations with only a slight slowdown.

Cross References

- ▶ Compressed Suffix Array
- ▶ Compressed Text Indexing
- ▶ Rank and Select Operations on Binary Strings
- ▶ Succinct Encoding of Permutations: Applications to Text Indexing
- ▶ Text Indexing

Recommended Reading

1. Bernhart, F., Kainen P.C.: The book thickness of a graph. J. Comb. Theory B **27**(3), 320–331 (1979)
2. Chiang, Y.-T., Lin, C.-C., Lu, H.-I.: Orderly spanning trees with applications. SIAM J. Comput. **34**(4), 924–945 (2005)
3. Chuang, R.C.-N., Garg, A., He, X., Kao, M.-Y., Lu, H.-I.: Compact encodings of planar graphs via canonical orderings and multiple parentheses. Comput. Res. Repos. cs.DS/0102005 (2001)
4. Geary, R.F., Rahman, N., Raman, R., Raman, V.: A simple optimal representation for balanced parentheses. Theor. Comput. Sci. **368**(3), 231–246 (2006)
5. Grossi, R., Gupta, A., Vitter J.S.: High-order entropy-compressed text indexes. In: Farach-Colton, M. (ed) Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, pp. 841–850, Philadelphia (2003)
6. Jacobson, G.: Space-efficient static trees and graphs. In: Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science, IEEE, pp. 549–554, New York (1989)
7. Lu, H.-I., Yeh, C.-C.: Balanced parentheses strike back. Accepted to ACM Trans. Algorithms (2007)
8. Munro, J.I., Raman V.: Succinct representation of balanced parentheses and static trees. SIAM J. Comput. **31**(3), 762–776 (2001)
9. Munro, J.I., Raman, V., Rao, S.S.: Space efficient suffix trees. J. Algorithms **39**(2), 205–222 (2001)
10. Munro, J.I., Raman, V., Storm, A.J.: Representing dynamic binary trees succinctly. In: Rao Kosaraju, S. (ed.) Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, pp. 529–536, Philadelphia (2001)
11. Munro, J.I., Rao, S.S.: Succinct representations of functions. In: Díaz, J., Karhumäki, J., Lepistö, A., Sannella, D. (eds.): Proceedings of the 31st International Colloquium on Automata, Languages and Programming, pp. 1006–1015. Springer, Heidelberg (2004)
12. Raman, R., Rao, S. S.: Succinct dynamic dictionaries and trees. In: Baeten, J.C.M., Lenstra, J.K., Parrow J., Woeginger, G.J. (eds.) Proceedings of the 30th International Colloquium on Automata, Languages and Programming, pp. 357–368. Springer, Heidelberg (2003)
13. Sadakane, K.: Compressed suffix trees with full functionality. Theory Comput. Syst. (2007) Online first. http://dx.doi.org/10.1007/s00224-006-1198-x
14. Weiner, P.: Linear pattern matching algorithms. In: Proceedings of the 14th Annual IEEE Symposium on Switching and Automata Theory, pp. 1–11. IEEE, New York (1973)
15. Yannakakis, M.: Four pages are necessary and sufficient for planar graphs. In: Hartmanis, J. (ed.) Proceedings of the 18th Annual ACM-SIAM Symposium on Theory of Computing, pp. 104–108. ACM, New York (1986)

Succinct Encoding of Permutations: Applications to Text Indexing

2003; Munro, Raman, Raman, Rao

JÉRÉMY BARBAY[1], J. IAN MUNRO[2]

[1] Department of Computer Science, University of Chile, Santiago, Chile

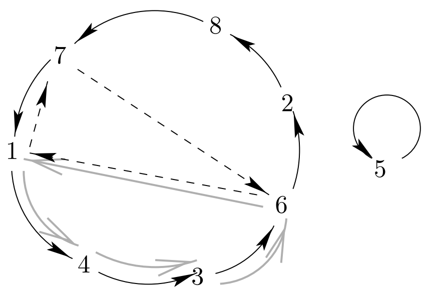
[2] Cheriton School of Computer Science, University of Waterloo, Waterloo, ON, Canada

Problem Definition

A succinct data structure for a given data type is a representation of the underlying combinatorial object that uses an amount of space "close" to the information theoretic lower bound together with algorithms that support operations of the data type "quickly." A natural example is the representation of a binary tree [5]: an arbitrary binary tree on n nodes can be represented in $2n + o(n)$ bits while supporting a variety of operations on any node, which include finding its parent, its left or right child, and returning the size of its subtree, each in $O(1)$ time. As there are $\binom{2n}{n}/(n+1)$ binary trees on n nodes and the logarithm of this term[1] is $2n - o(n)$, the space used by this representation is optimal to within a lower-order term.

In the applications considered in this entry, the principle concern is with indexes supporting search in strings and in XML-like documents (i. e., tree-structured objects with labels and "free text" at various nodes). As it happens, not only labeled trees but also arbitrary binary relations

[1] All logarithms are taken to the base 2. By convention, the iterated logarithm is denoted by $\lg^{(i)} n$; hence, $\lg\lg\lg x$ is $\lg^{(3)} x$.



i		1	2	3	4	5	6	7	8
$\pi(i)$	=	4	8	6	3	5	2	1	7
B	=	1	0	0	0	0	1	1	0
S	=	7					1	6	

Succinct Encoding of Permutations: Applications to Text Indexing, Figure 1
A permutation on $\{1, \ldots, 8\}$, with two cycles and three back pointers. The *full black lines* correspond to the permutation, the *dashed lines* to the back pointers and the *gray lines* to the edges traversed to compute $\pi^{-1}(3)$

over finite domains are key building blocks for this. Preprocessing such data structures so as to be able to perform searches is a complex process requiring a variety of subordinate structures.

A basic building block for this work is the representation of a permutation of the integers $\{0, \ldots, n-1\}$, denoted by $[n]$. A permutation π is trivially representable in $n\lceil\lg n\rceil$ bits which is within $O(n)$ bits of the information theoretic bound of $\lg(n!)$. The interesting problem is to support both the permutation and its inverse: namely, how to represent an arbitrary permutation π on $[n]$ in a succinct manner so that $\pi^k(i)$ (π iteratively applied k times starting at i, where k can be any integer so that π^{-1} is the inverse of π) can be evaluated quickly.

Key Results

Munro et al. [7] studied the problem of succinctly representing a permutation to support computing $\pi^k(i)$ quickly. They give two solutions: one supports the operations arbitrarily quickly, at the cost of extra space; the other uses essentially optimal space at the cost of slower evaluation.

Given an integer parameter t, the permutations π and π^{-1} can be supported by simply writing down π in an array of n words of $\lceil\lg n\rceil$ bits each, plus an auxiliary array S of at most n/t shortcuts or back pointers. In each cycle of length at least t, every tth element has a pointer t steps back. $\pi(i)$ is simply the ith value in the primary structure, and $\pi^{-1}(i)$ is found by moving forward until a back pointer is found and then continuing to follow the cycle to the location that contains the value i. The trick is in the encoding of the locations of the back pointers: this is done with a simple bit vector B of length n, in which a 1 indicates that a back pointer is associated with a given location. B is augmented using $o(n)$ additional bits so that the number of 1's up to a given position and the position of the rth 1 can be found in constant time (i. e., using the rank and select operations on binary strings [8]). This gives the location of the appropriate back pointer in the auxiliary array S.

For example, the permutation $\pi = (4, 8, 6, 3, 5, 2, 1, 7)$ consists of two cycles, $(\mathbf{1}, 4, 3, \mathbf{6}, 2, 8, \mathbf{7})$ and (5) (Fig. 1). For $t = 3$, the back pointers are cycling backward between 1, 6 and 7 in the largest cycle (there are none in the other because it is smaller than t). To find $\pi^{-1}(3)$, follow π from 3 to 6, observe that 6 is a back pointer because it is marked by the second 1 in B, and follow the second value of S to 1, then follow π from 1 to 4 and then to 3: the predecessor of 3 has been found. As there are back pointers every t elements in the cycle, finding the predecessor requires $O(t)$ memory accesses.

For arbitrary i and k, $\pi^k(i)$ is supported by writing the cycles of π together with a bit vector B marking the beginning of each cycle. Observe that the cycle representation itself is a permutation in "standard form," call it σ. For example, the permutation $\pi = (6, 4, 3, 5, 2, 1)$ has three cycles $\{(1, 6), (3), (2, 5, 4)\}$ and is encoded by the permutation $\sigma = (1, 6, 3, 2, 5, 4)$ and the bit vector $B = (1, 0, 1, 1, 0, 0)$. The first task is to find i in the representation: it is in position $\sigma^{-1}(i)$. The segment of the representation containing i is found through the rank and select operations on B. From this $\pi^k(i)$ is easily determined by taking k modulo the cycle length and moving that number of steps around the cycle starting at the position of i.

Other than the support of the inverse of σ, all operations are performed in constant time; hence, the total time depends on the value chosen for t.

Theorem 1 (Munro et al. [7]) *There is a representation of an arbitrary permutation π on $[n]$ using at most $(1+\varepsilon)n\lg n + O(n)$ bits that can support the operation $\pi^k()$ in time $O(1/\varepsilon)$, for any constant ε less than 1 and for any arbitrary value of k.*

It is not difficult to prove that this technique is optimal under a restricted model of a pointer machine. So, for example, using $O(n)$ extra bits (i. e., $O(n/\lg n)$ extra words), $\Omega(\lg n)$ time is necessary to compute both π and π^{-1}. However, using another approach Munro et al. [7] demonstrated that the lower bound suggested does not hold in the RAM model. The approach is based on the Benes network, a communication network composed of switches that can be used to implement permutations.

Theorem 2 (Munro et al. [7]) *There is a representation of an arbitrary permutation π on $[n]$ using at most $\lceil \lg(n!) \rceil + O(n)$ bits that can support the operation $\pi^k()$ in time $O(\lg n/\lg^{(2)} n)$.*

While this data structure uses less space than the other, it requires more time for each operation. It is not known whether this time bound can be improved using only $O(n)$ "extra space." As a consequence, the first data structure is used in all applications. Obviously, any other solution can be used, potentially with a better time/space trade-off.

Applications

The results on permutations are particularly useful for two lines of research: first in the extension of the results on permutation to arbitrary integer functions; and second, and probably more importantly, in encoding and indexing text strings, which themselves are used to encode sparse binary relations and labeled trees. This section summarizes some of these results.

Functions

Munro and Rao [9] extended the results on permutations to arbitrary functions from $[n]$ to $[n]$. Again $f^k(i)$ indicates the function iterated k times starting at i. If k is nonnegative, this is straightforward. The case in which k is negative is more interesting as the image is a (possibly empty) multiset over $[n]$ (see Fig. 2 for an example). Whereas π is a set of cycles, f can be viewed as a set of cycles in which each node is the root of a tree. Starting at any node (element of $[n]$), the evaluation moves one step toward the root of the tree or one step along a cycle (e. g., $f(8) = 7$, $f(10) = 11$). Moving k steps in a positive direction is straightforward; one moves up a tree and perhaps around a cycle (e. g. $f^5(9) = f^3(9) = 3$) When k is negative one must determine all nodes of distance k from the starting location, i, in the direction towards the leaves of the trees (e. g., $f^{-1}(13) = \{1, 11, 12\}$, $f^{-1}(3) = \{4, 5\}$). The key technical issue is to run across succinct tree representations picking off all nodes at the appropriate levels.

Theorem 3 (Munro and Rao [9]) *For any fixed ε, there is a representation of a function $f\colon [n] \to [n]$ that takes $(1+\varepsilon)n \lg n + O(1)$ bits of space, and supports $f^k(i)$ in $O(1 + |f^k(i)|)$ time, for any integer k and for any $i \in [n]$.*

Text Strings

Indexing text strings to support the search for patterns is an important general issue. Barbay et al. [2] considered "negative" searches, along the following lines.

Definition 1 Consider a string $S[1, n]$ over the alphabet $[l]$. A position $x \in [n]$ *matches a literal* $\alpha \in [l]$ if $S[x] = \alpha$. A position $x \in [n]$ *matches a literal* $\bar{\alpha}$ if $S[x] \neq \alpha$. The set $\{\bar{1}, \ldots, \bar{l}\}$ is denoted by $[\bar{l}]$.

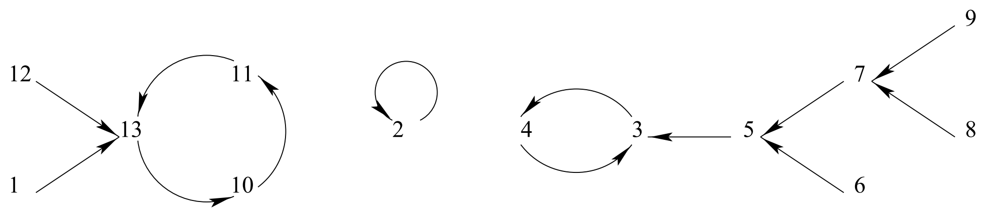
Given a string S of length n over an alphabet of size l, for any position x in the string, any literal $\alpha \in [l] \cup [\bar{l}]$ and any integer r, consider the following operators:

- string_rank$_S(\alpha, x)$: the number of occurrences of α in $S[1..x]$;
- string_select$_S(\alpha, r)$: the position of the rth occurrence of α in S, or ∞ if none exists;
- string_access$_S(x)$: the label $S[x]$;
- string_pred$_S(\alpha, x)$: the last occurrence of α in $S[1 \ldots x]$, or ∞ if none exists;
- string_succ$_S(\alpha, r)$: the first occurrence of α in $S[x \ldots]$, or ∞ if none exists.

Golynski et al. [4] observed that a string of length l on alphabet $[l]$ can be encoded and indexed by a permutation on $[l]$ (which for each label lists the positions of all its occurrences) together with a bit vector of length $2l$ (which signals the end of each sublist of occurrences corresponding to a label). For instance, the string $ACCA$ on alphabet $\{A, B, C, D\}$ is encoded by the permutation $(1, 4, 2, 3)$ and the bit vector $(0, 0, 1, 1, 1, 0, 0, 1)$. Golynski et al. were then able to support the operators rank, select and access in time $O(\lg^{(2)} n)$, by using a value of $t = \lg^{(2)} n$ in the encoding of permutation of Theorem 1.

This encoding achieves fast support for the search operators defined above restricted to labels (not literals), with a small overhead in space, by integrating the encodings of the text and the indexing information. Barbay et al. [2] extended those operators to literals, and showed how to separate the *succinct encoding* of the string S, in a manner that assumes we can access a word of S in a fixed time bound, and a *succinct index* containing auxiliary information useful to support the search operators defined above.

Theorem 4 (Barbay et al. [2]) *Given access to a label in the raw encoding of a string $S \in [l]^n$ in time $f(n, l)$, there is a succinct index using $n(1 + o(\lg l))$ bits that supports the operators* string_rank$_S$, string_pred$_S$



i	1	2	3	4	5	6	7	8	9	10	11	12	13
$f(i) =$	13	2	4	3	3	5	5	7	7	11	13	13	10
$f^{-1}(i) =$	{}	{2}	{4,5}	{3}	{6,7}	{}	{8,9}	{}	{}	{13}	{10}	{}	{1,11,12}

Succinct Encoding of Permutations: Applications to Text Indexing, Figure 2
A function on $\{1,\ldots,13\}$, with three cycles and two nontrivial tree structures

and string_succ$_S$ *for any literal* $\alpha \in [l] \cup [\bar{l}]$ *in* $O(\lg^{(2)} l \cdot \lg^{(3)} l \cdot (f(n,t) + \lg^{(2)} l))$ *time, and the operator* string_select$_S$ *for any label* $\alpha \in [l]$ *in* $O(\lg^{(3)} l \cdot (f(n,t) + \lg^{(2)} l))$ *time.*

The separation between the encoding of the string or of an XML-like document and its index has two main advantages:

1. The string can now be compressed and searched at the same time, provided that the compressed encoding of the string supports the access in reasonable time, as does the one described by Ferragina and Venturini [3].
2. The operators can be supported for several orderings of the string, for instance, induced by distinct traversals of a labeled tree, with only a small cost in space. It is important, for instance, when those orders correspond to various traversals of a labeled structure, such as the depth-first and Depth First Uniary Degree Sequence (DFUDS) traversals of a labeled tree [2].

Binary Relations

Given two ordered sets of sizes l and n, denoted by $[l]$ and $[n]$, a binary relation R between these sets is a subset of their Cartesian product, i. e., $R \subset [l]\times[n]$. It is used, for instance, to represent the relation between a set of labels $[l]$ and a set of objects $[n]$.

Although a string can be seen as a particular case of a binary relation, where the objects are positions and exactly one label is associated with each position, the search operations on binary relations are diverse, including operators on both the labels and the objects. For any literal α, object x and integer r, consider the following operators:

- label_rank$_R(\alpha, x)$: the number of objects labeled α preceding or equal to x;
- label_select$_R(\alpha, r)$: the position of the rth object labeled α if any, or ∞ otherwise;
- label_nb$_R(\alpha)$, the number of objects with label α;
- object_rank$_R(x, \alpha)$: the number of labels associated with object x preceding or equal to label α;
- object_select$_R(x, r)$: the rth label associated with object x, if any, or ∞ otherwise;
- object_nb$_R(x)$: the number of labels associated with object x;
- table_access$_R(x, \alpha)$: checks whether object x is associated with label α.

Barbay et al. [1] observed that such a binary relation, consisting of t pairs from $[n] \times [l]$, can be encoded as a text string S listing the t labels, and a binary string B indicating how many labels are associated with each object. So search operations on the objects associated with a fixed label are reduced to a combination of operators on text and binary strings. Using a more direct reduction to the encoding of permutations, the index of the binary relation can be separated from its encoding, and even more operators can be supported [2].

Theorem 5 (Barbay et al. [2]) *Given support for* object_access$_R$ *in* $f(n,l,t)$ *time on a binary relation formed by* t *pairs from an object set* $[n]$ *and a label set* $[l]$, *there is a succinct index using* $t(1 + o(\lg l))$ *bits that supports* label_rank$_R$ *for any literal* $\alpha \in [l] \cup [\bar{l}]$ *and* label_access$_R$ *for any label* $\alpha \in [l]$ *in* $O(\lg^{(2)} l \cdot \lg^{(3)} l \cdot (f(n,l,t) + \lg^{(2)} l))$ *time, and* label_select$_R$ *for any label* $\alpha \in [l]$ *in* $O(\lg^{(3)} l \cdot (f(n,l,t) + \lg^{(2)} l))$ *time.*

To conclude this entry, note that a labeled tree T can be represented by an ordinal tree coding its structure [6] and a string S listing the labels of the nodes. If the labels are listed in preorder (respectively in DFUDS order) the operator string_succ$_S$ enumerates all the descendants (respectively children) of a node matching some literal α. Using succinct indexes, a single encoding of the labels and the support of a permutation between orders is sufficient to implement both enumerations, and other search operators on the labels. These issues, along with strings and labeled trees compression techniques which achieve the entropy of the indexed data, are covered in more detail in the entries cited in ► Tree Compression and Indexing.

Cross References

► Compressed Suffix Array
► Compressed Text Indexing
► Rank and Select Operations on Binary Strings
► Text Indexing

Recommended Reading

1. Barbay, J., Golynski, A., Munro, J.I., Rao, S.S.: Adaptive searching in succinctly encoded binary relations and tree-structured documents. In: Proceedings of the 17th Annual Symposium on Combinatorial Pattern Matching (CPM). Lecture Notes in Computer Science (LNCS), vol. 4009, pp. 24–35. Springer, Berlin (2006)
2. Barbay, J., He, M., Munro, J.I., Rao, S.S.: Succinct indexes for strings, binary relations and multi-labeled trees. In: Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 680–689. ACM, SIAM (2007)
3. Ferragina, P., Venturini, R.: A simple storage scheme for strings achieving entropy bounds. In: Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 690–695. ACM, SIAM (2007)
4. Golynski, A., Munro, J.I., Rao, S.S.: Rank/select operations on large alphabets: a tool for text indexing. In: Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 368–373. ACM, SIAM (2006)
5. Jacobson, G.: Space-efficient static trees and graphs. In: Proceedings of the 30th IEEE Symposium on Foundations of Computer Science (FOCS), pp. 549–554 (1989)
6. Jansson, J., Sadakane, K., Sung, W.-K.: Ultra-succinct representation of ordered trees. In: Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 575–584. ACM, SIAM (2007)
7. Munro, J.I., Raman, R., Raman, V., Rao, S.S.: Succinct representations of permutations. In: Proceedings of the 30th International Colloquium on Automata, Languages and Programming (ICALP). Lecture Notes in Computer Science (LNCS), vol. 2719, pp. 345–356. Springer, Berlin (2003)
8. Munro, J.I., Raman, V.: Succinct representation of balanced parentheses and static trees. SIAM J. Comput. **31**, 762–776 (2001)
9. Munro, J.I., Rao, S.S.: Succinct representations of functions. In: Proceedings of the International Colloquium on Automata, Languages and Programming (ICALP). Lecture Notes in Computer Science (LNCS), vol. 3142, pp. 1006–1015. Springer, Berlin (2004)

Suffix Array Construction

2006; Kärkkäinen, Sanders, Burkhardt

JUHA KÄRKKÄINEN
Department of Computer Science, University of Helsinki, Helsinki, Finland

Keywords and Synonyms

Suffix sorting; Full-text index construction

Problem Definition

The *suffix array* [5,14] is the lexicographically sorted array of all the suffixes of a string. It is a popular text index structure with many applications. The subject of this entry are algorithms that construct the suffix array.

More precisely, the input to a suffix array construction algorithm is a *text string* $T = T[0, n) = t_0 t_1 \cdots t_{n-1}$, i. e., a sequence of n *characters* from an *alphabet* Σ. For $i \in [0, n]$, let S_i denote the *suffix* $T[i, n) = t_i t_{i+1} \cdots t_{n-1}$. The output is the *suffix array* $SA[0, n]$ of T, a permutation of $[0, n]$ satisfying $S_{SA[0]} < S_{SA[1]} < \cdots < S_{SA[n]}$, where $<$ denotes the *lexicographic order* of strings.

Two specific models for the alphabet Σ are considered. An *ordered alphabet* is an arbitrary ordered set with constant time character comparisons. An *integer alphabet* is the integer range $[1, n]$. There is also a result that holds for any alphabet.

Many applications require that the suffix array is augmented with additional information, most commonly with the *longest common prefix* array $LCP[0, n)$. An entry $LCP[i]$ of the LCP array is the length of the longest common prefix of the suffixes $S_{SA[i]}$ and $S_{SA[i+1]}$. The *enhanced suffix array* [1] adds two more arrays to obtain a full range of text index functionalities.

Another related array, the *Burrows–Wheeler transform* $BWT[0, n)$ is often computed by suffix array construction using the equations $BWT[i] = T[SA[i] - 1]$ when $SA[i] \neq 0$ and $BWT[i] = T[n-1]$ when $SA[i] = 0$.

There are other important text indexes, most notably suffix trees and compressed text indexes, covered in separate entries. Each of these indexes have their own construction algorithms, but they can also be constructed efficiently from each other. However, in this entry, the focus is on direct suffix array construction algorithms that do not rely on other text indexes.

Key Results

The naive approach to suffix array construction is to use a general sorting algorithm or an algorithm for sorting strings. However, any such algorithm has a worst-case time complexity $\Omega(n^2)$ because the total length of the suffixes is $\Omega(n^2)$.

The first efficient algorithms were based on the *doubling technique* of Karp, Miller, and Rosenberg [8]. The idea is to assign a *rank* to all substrings whose length is a power of two. The rank tells the lexicographic order of the substring among substrings of the same length. Given the ranks for substrings of length h, the ranks for substrings of length $2h$ can be computed using a radixsort step in linear time (doubling). The technique was first applied to suffix array construction by Manber and Myers [14]. The best practical algorithm based on the technique is by Larsson and Sadakane [13].

Theorem 1 (Manber and Myers [14]; Larsson and Sadakane [13]) *The suffix array can be constructed in $\mathcal{O}(n \log n)$ worst-case time, which is optimal for the ordered alphabet.*

Faster algorithms for the integer alphabet are based on a different technique, recursion. The basic procedure is as follows.

1. Sort a subset of the suffixes. This is done by constructing a shorter string, whose suffix array gives the order of the desired subset. The suffix array of the shorter string is constructed by recursion.
2. Extend the subset order to full order.

The technique first appeared in suffix tree construction [4], but 2003 saw the independent and simultaneous publication of three linear time suffix array construction algorithms based on the approach but not using suffix trees. Each of the three algorithms uses a different subset of suffixes requiring a different implementation of the second step.

Theorem 2 (Kärkkäinen, Sanders and Burkhardt [7]; Kim el al. [10]; Ko and Aluru [11]) *The suffix array can be constructed in the optimal linear time for the integer alphabet.*

The algorithm of Kärkkäinen, Sanders, and Burkhardt [7] has generalizations for several parallel and hierarchical memory models of computation including an optimal algorithm for external memory and a linear work algorithm for the BSP model.

The above algorithms and many other suffix array construction algorithms are surveyed in [18].

The $\Omega(n \log n)$ lower bound for the ordered alphabet mentioned in Theorem 1 comes from the sorting complexity of characters, since the initial characters of the sorted suffixes are the text characters in sorted order. Theorem 2 allows a generalization of this result. For any alphabet, one can first sort the characters of T, remove duplicates, assign a rank to each character, and construct a new string T' over the alphabet $[1, n]$ by replacing the characters of T with their ranks. The suffix array of T' is exactly the same as the suffix array of T. Optimal algorithms for the integer alphabet then give the following result.

Theorem 3 *For any alphabet, the complexity of suffix array construction is the same as the complexity of sorting the characters of the string.*

The result extends to the related arrays.

Theorem 4 (Kasai et al. [9]; Abouelhoda, Kurtz and Ohlebusch [1]) *The LCP array, the enhanced suffix array, and the BWT can be computed in linear time given the suffix array.*

One of the main advantages of suffix arrays over suffix trees is their smaller space requirement (by a constant factor), and a significant effort has been spent making construction algorithms space efficient, too. A technique based on the notion of *difference covers* gives the following results.

Theorem 5 (Burkhardt and Kärkkäinen [2]; Kärkkäinen, Sanders and Burkhardt [7]) *For any $v = \mathcal{O}(n^{2/3})$, the suffix array can be constructed in $\mathcal{O}(n(v + \log n))$ time for the ordered alphabet and in $\mathcal{O}(nv)$ time for the integer alphabet using $\mathcal{O}(n/\sqrt{v})$ space in addition to the input (the string T) and the output (the suffix array).*

Kärkkäinen [6] uses the difference cover technique to construct the suffix array in blocks without ever storing the full suffix array obtaining the following result for computing the BWT.

Theorem 6 (Kärkkäinen [6]) *For any $v = \mathcal{O}(n^{2/3})$, the BWT can be constructed in $\mathcal{O}(n(v + \log n))$ time for the ordered alphabet using $\mathcal{O}(n/\sqrt{v})$ space in addition to the input (the string T) and the output (the BWT).*

Compressed text index construction algorithms are alternatives to space-efficient BWT computation.

Applications

The suffix array is a simple and powerful text index structure with numerous applications detailed in the entry *Text Indexing*. In addition, due to the existence of efficient and practical construction algorithms, the suffix array is often used as an intermediate data structure in computing something else. The BWT is usually computed from the suffix array and has applications in text compression and compressed index construction. The suffix tree is also easy to construct given the suffix array and the LCP array.

Open Problems

Theoretically, the suffix array construction problem is essentially solved. The development of ever more efficient practical algorithms is still going on with several different nontrivial heuristics available [18] including very recent ones [15].

Experimental Results

An experimental comparison of a large number of suffix array construction algorithms is presented in [18]. The best algorithms in the comparison are the algorithm by Maniscalco and Puglisi [15], which is the fastest but has an $\Omega(n^2)$ worst-case complexity, and a variant of the algorithm by Burkhardt and Kärkkäinen [2], which is the fastest among algorithms with good worst-case complexity. Both algorithms are also space efficient. The algorithm of Manzini and Ferragina [17] is still slightly more space efficient and also very fast in practice.

There are also experiments with parallel [12] and external memory algorithms [3]. Variants of the algorithm by Kärkkäinen, Sanders and Burkhardt [7] show high performance and scalability in both cases.

Algorithms for computing the LCP array from the suffix array are compared in [16].

Data Sets

The input to a suffix array construction algorithm is simply a text, so an abundance of data exists. Commonly used text collections include the Canterbury Corpus at http://corpus.canterbury.ac.nz/, the corpus compiled by Manzini and Ferragina at http://www.mfn.unipmn.it/~manzini/lightweight/corpus/, and the Pizza&Chili Corpus at http://pizzachili.dcc.uchile.cl/texts.html.

URL to Code

The implementations of many of the algorithms mentioned here are publicly available, for example: http://www.larsson.dogma.net/research.html [13], http://www.mpi-sb.mpg.de/~sanders/programs/suffix/ [7], and http://www.cs.helsinki.fi/juha.karkkainen/publications/cpm03.tar.gz [2]. Manzini provides a package that computes the LCP array and the BWT, too, at http://www.mfn.unipmn.it/~manzini/lightweight/index.html. The `bzip2` compression program (http://www.bzip.org/) computes the BWT through suffix array construction.

Cross References

- ► Compressed Suffix Array
- ► Compressed Text Indexing
- ► String Sorting
- ► Suffix Tree Construction in Hierarchical Memory
- ► Suffix Tree Construction in RAM
- ► Text Indexing
- ► Two-Dimensional Pattern Indexing

Recommended Reading

1. Abouelhoda, M.I., Kurtz, S., Ohlebusch, E.: Replacing suffix trees with enhanced suffix arrays. J. Discret. Algorithms **2**, 53–86 (2004)
2. Burkhardt, S., Kärkkäinen, J.: Fast lightweight suffix array construction and checking. In: Proc. 14th Annual Symposium on Combinatorial Pattern Matching. LNCS, vol. 2676, pp. 55–69. Springer, Berlin/Heidelberg (2003)
3. Dementiev, R., Mehnert, J., Kärkkäinen, J., Sanders, P.: Better external memory suffix array construction. ACM J. Exp. Algorithmics (2008) in press
4. Farach-Colton, M., Ferragina, P., Muthukrishnan, S.: On the sorting-complexity of suffix tree construction. J. Assoc. Comput. Mach. **47**, 987–1011 (2000)
5. Gonnet, G., Baeza-Yates, R., Snider, T.: New indices for text: PAT trees and PAT arrays. In: Frakes, W.B., Baeza-Yates, R. (eds.) Information Retrieval: Data Structures & Algorithms. pp. 66–82 Prentice-Hall, Englewood Cliffs (1992)
6. Kärkkäinen, J.: Fast BWT in small space by blockwise suffix sorting. Theor. Comput. Sci. **387**, 249–257 (2007)
7. Kärkkäinen, J., Sanders, P., Burkhardt, S.: Linear work suffix array construction. J. Assoc. Comput. Mach. **53**, 918–936 (2006)
8. Karp, R.M., Miller, R.E., Rosenberg, A.L.: Rapid identification of repeated patterns in strings, trees and arrays. In: Proc. 4th Annual ACM Symposium on Theory of Computing, pp. 125–136. ACM Press, New York (1972)
9. Kasai, T., Lee, G., Arimura, H., Arikawa, S., Park, K.: Linear-time longest-common-prefix computation in suffix arrays and its applications. In: Proc. 12th Annual Symposium on Combinatorial Pattern Matching, vol. (2089) of LNCS. pp. 181–192. Springer, Berlin/Heidelberg (2001)
10. Kim, D.K., Sim, J.S., Park, H., Park, K.: Constructing suffix arrays in linear time. J. Discret. Algorithms **3**, 126–142 (2005)
11. Ko, P., Aluru, S.: Space efficient linear time construction of suffix arrays. J. Discret. Algorithms **3**, 143–156 (2005)
12. Kulla, F., Sanders, P.: Scalable parallel suffix array construction. In: Proc. 13th European PVM/MPI User's Group Meeting. LNCS, vol. 4192, pp. 22–29. Springer, Berlin/Heidelberg (2006)
13. Larsson, N.J., Sadakane, K.: Faster suffix sorting. Theor. Comput. Sci. **387**, 258–272 (2006)
14. Manber, U., Myers, G.: Suffix arrays: A new method for on-line string searches. SIAM J. Comput. **22**, 935–948 (1993)
15. Maniscalco, M.A., Puglisi, S.J.: Faster lightweight suffix array construction. In: Proc. 17th Australasian Workshop on Combinatorial Algorithms, pp. 16–29. Univ. Ballavat, Ballavat (2006)
16. Manzini, G.: Two space saving tricks for linear time LCP array computation. In: Proc. 9th Scandinavian Workshop on Algorithm Theory. LNCS, vol. 3111, pp. 372–383. Springer, Berlin/Heidelberg (2004)
17. Manzini, G., Ferragina, P.: Engineering a lightweight suffix array construction algorithm. Algorithmica **40**, 33–50 (2004)
18. Puglisi, S., Smyth, W., Turpin, A.: A taxonomy of suffix array construction algorithms. ACM Comput. Surv. **39**(2), Article 4, 31 pages (2007)

Suffix Tree Construction in Hierarchical Memory

2000; Farach-Colton, Ferragina, Muthukrishnan

PAOLO FERRAGINA
Department of Computer Science, University of Pisa, Pisa, Italy

Keywords and Synonyms

Suffix array construction; String B-tree construction; Full-text index construction

Problem Definition

The suffix tree is the ubiquitous data structure of combinatorial pattern matching because of its elegant uses in a myriad of situations–-just to cite a few, searching, data compression and mining, bioinformatics [6]. In these applications, the large data sets now available involve the use of numerous memory levels which constitute the storage medium of modern PCs: L1 and L2 caches, internal memory, multiple disks and remote hosts over a network. The power of this memory organization is that it may be able to offer the expected access time of the fastest level (i. e. cache) while keeping the average cost per memory cell near the one of the cheapest level (i. e. disk), provided that data are properly *cached* and *delivered* to the requiring algorithms. Neglecting questions pertaining to the cost of memory references may even prevent the use of algorithms on large sets of input data. Engineering research is presently trying to improve the input/output subsystem to reduce the impact of these issues, but it is very well known [16] that the improvements achievable by means of a *proper arrangement of data* and a properly *structured algorithmic computation* abundantly surpass the best-expected technology advancements.

The Model of Computation

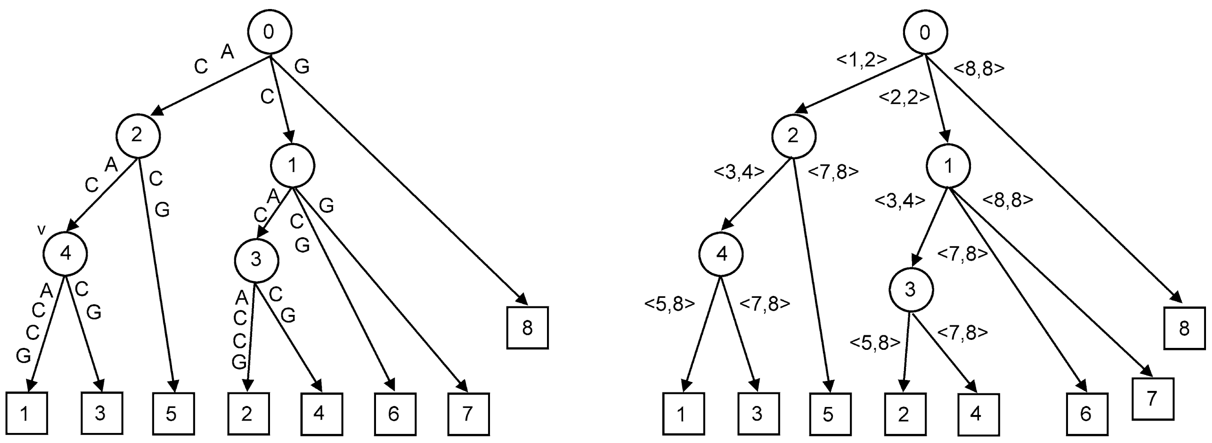
In order to reason about algorithms and data structures operating on hierarchical memories, it is necessary to introduce a *model of computation* that grasps the essence of real situations so that algorithms that are good in the model are also good in practice. The model considered here is the *external memory model* [16], which received much attention because of its simplicity and reasonable accuracy. A computer is abstracted to consist of *two memory levels*: the internal memory of size M, and the (unbounded) disk memory which operates by reading/writing data in blocks of size B (called *disk pages*). The performance of algorithms is then evaluated by counting: (a) the number of disk accesses (I/Os), (b) the internal running time (CPU time), and (c) the number of disk pages occupied by the data structure or used by the algorithm as its working space. This simple model suggests, correctly, that a good external-memory algorithm should exploit both *spatial locality* and *temporal locality*. Of course, “I/O” and “two-level view” refer to any two levels of the memory hierarchy with their parameters M and B properly set.

Notation

Let $S[1, n]$ be a string drawn from alphabet Σ, and consider the notation: S_i for the ith *suffix* of string S, $\mathtt{lcp}(\alpha, \beta)$ for the *longest common prefix* between the two strings α and β, and $\mathtt{lca}(u, v)$ for the *lowest common ancestor* between two nodes u and v in a tree.

The suffix tree of $S[1, n]$, denoted hereafter by $\mathcal{T}_S$, is a tree that stores all suffixes of $S\#$ in a compact form, where $\# \notin \Sigma$ is a special character (see Fig. 1). $\mathcal{T}_S$ consists of n leaves, numbered from 1 to n, and any root-to-leaf path spells out a suffix of $S\#$. The endmarker # guarantees that no suffix is the prefix of another suffix in $S\#$. Each internal node has at least two children and each edge is labeled with a non empty substring of S. No two edges out of a node can begin with the same character, and sibling edges are ordered lexicographically according to that character. Edge labels are encoded with pairs of integers – say $S[x, y]$ is represented by the pair $\langle x, y\rangle$. As a result, all $\Theta(n^2)$ substrings of S can be represented in $O(n)$ optimal space by $\mathcal{T}_S$'s structure and edge encoding. Furthermore, the rightward scan of the suffix tree leaves gives the ordered set of S's suffixes, also known as the *suffix array* of S [12]. Notice that the case of a large string collection $\Delta = \{S^1, S^2, \ldots, S^k\}$ *reduces to* the case of one long string $S = S^1\#_1 S^2\#_2 \cdots S^k\#_k$, where $\#_i \notin \Sigma$ are special symbols.

Numerous algorithms are known that build the suffix tree optimally in the RAM model (see [3] and references therein). However, most of them exhibit a marked absence of *locality of references* and thus elicit many I/Os when the size of the indexed string is too large to be fit into the internal memory of the computer. This is a serious problem because the slow performance of these algorithms can prevent the suffix tree being used even in medium-scale applications. This encyclopedia's entry surveys algorithmic solutions that deal efficiently with the *construction of suffix trees over large string collections* by executing an optimal number of I/Os. Since it is assumed that the edges leaving a node in $\mathcal{T}_S$ are lexicographically sorted, sorting is an obvious *lower bound* for building suffix trees (consider the suffix tree of a permutation!). The presented algorithms



Suffix Tree Construction in Hierarchical Memory, Figure 1
The suffix tree of S = ACACACCG on the left, and its compact edge-encoding on the right. The endmarker # is not shown. Node v spells out the string ACAC. Each internal node stores the length of its associated string, and each leaf stores the starting position of its corresponding suffix

```
DIVIDE-AND-CONQUER ALGORITHM
    (1) Construct the string $S'[j]$ = rank of $\langle S[2j], S[2j+1]\rangle$, and recursively compute $\mathcal{T}_{S'}$.
    (2) Derive from $\mathcal{T}_{S'}$ the compacted trie $\mathcal{T}_o$ of all suffixes of $S$ beginning at odd positions.
    (3) Derive from $\mathcal{T}_o$ the compacted trie $\mathcal{T}_e$ of all suffixes of $S$ beginning at even positions.
    (4) Merge $\mathcal{T}_o$ and $\mathcal{T}_e$ into the whole suffix tree $\mathcal{T}_S$, as follows:
        (4.1) Overmerge $\mathcal{T}_o$ and $\mathcal{T}_e$ into the tree $\mathcal{T}_M$.
        (4.2) Partially unmerge $\mathcal{T}_M$ to get $\mathcal{T}_S$.
```

Suffix Tree Construction in Hierarchical Memory, Figure 2
The algorithm that builds the suffix tree directly

have sorting as their bottleneck, thus establishing that *the complexity of sorting and suffix tree construction match.*

Key Results

Designing a disk-efficient approach to suffix-tree construction has found efficient solutions only in the last few years [4]. The present section surveys two theoretical approaches which achieve the best (optimal!) I/O-bounds in the worst case, the next section will discuss some practical solutions.

The first algorithm is based on a *Divide-and-Conquer* approach that allows us to reduce the construction process to external-memory sorting and few low-I/O primitives. It builds the suffix tree $\mathcal{T}_S$ by executing four (macro)steps, detailed in Fig. 2. It is not difficult to implement the first three steps in $\texttt{Sort}(n) = O(\frac{n}{B} \log_{M/B} \frac{n}{B})$ I/Os [16]. The last (merging) step is the most difficult one and its I/O-complexity bounds the cost of the overall approach. [3] proposes an elegant merge for $\mathcal{T}_o$ and $\mathcal{T}_e$: substep (4.1) temporarily relaxes the requirement of getting $\mathcal{T}_S$ in one shot, and thus it blindly (over)merges the paths of $\mathcal{T}_o$ and $\mathcal{T}_e$ by comparing edges only via their first characters; then substep (4.2) re-fixes $\mathcal{T}_M$ by detecting and undoing in an I/O-efficient manner the (over)merged paths. Note that the time and I/O-complexity of this algorithm follow a nice recursive relation: $T(n) = T(n/2) + O(\texttt{Sort}(n))$.

Theorem 1 (Farach-Colton et al. 1999) *Given an arbitrary string $S[1, n]$, its suffix tree can be constructed in $O(\texttt{Sort}(n))$ I/Os, $O(n \log n)$ time and using $O(n/B)$ disk pages.*

The second algorithm is deceptively simple, elegant and I/O-optimal, and applies successfully to the construction of other indexing data structures, like the String B-tree [5]. The key idea is to derive $\mathcal{T}_S$ from the suffix array $\mathcal{A}_S$ and from the lcp array, which stores the longest-common-prefix length of adjacent suffixes in $\mathcal{A}_S$. Its pseudocode is given in Fig. 3. Note that Step (1) may deploy any external-memory algorithm for suffix array construc-

```
SuffixArray-based algorithm
  (1) Construct the suffix array $\mathcal{A}_S$ and the array $\texttt{lcp}_S$ of the string $S$.
  (2) Initially set $\mathcal{T}_S$ as a single edge connecting the root to a leaf pointing to suffix $\mathcal{A}_S[1]$.
  (2) For $i = 2, \ldots, n$:
    (2.1) Create a new leaf $\ell_i$ that points to the suffix $\mathcal{A}_S[i]$.
    (2.2) Walk up from $\ell_{i-1}$ until a node $u_i$ is met whose string-length $x_i$ is $\leq \texttt{lcp}_S[i]$.
    (2.3) If $x_i = \texttt{lcp}_S[i]$, leaf $\ell_i$ is attached to $u_i$.
    (2.4) If $x_i < \texttt{lcp}_S[i]$, create node $u'_i$ with string-length $x_i$, attach it to $u_i$ and leaf $\ell_i$ to $u'_i$.
```

Suffix Tree Construction in Hierarchical Memory, Figure 3
The algorithm that builds the suffix tree passing through the suffix array

tion: Used here is the elegant and optimal *Skew* algorithm of [9] which takes $O(\texttt{Sort}(n))$ I/Os. Step (2) takes a total of $O(n/B)$ I/Os by using a stack that stores the nodes on the current rightmost path of $\mathcal{T}_S$ in reversed order, i. e. leaf ℓ_i is on top. Walking upward, splitting edges or attaching nodes in $\mathcal{T}_S$ boils down to popping/pushing nodes from this stack. As a result, the time and I/O-complexity of this algorithm follow the recursive relation: $T(n) = T(2n/3) + O(\texttt{Sort}(n))$.

Theorem 2 (Kärkkäinen and Sanders 2003) *Given an arbitrary string $S[1, n]$, its suffix tree can be constructed in $O(\texttt{Sort}(n))$ I/Os, $O(n \log n)$ time and using $O(n/B)$ disk pages.*

It is not evident which one of these two algorithms is better in practice. The first one exploits a recursion with parameter 1/2 but incurs a large space overhead because of the management of the tree topology; the second one is more space efficient and easier to implement, but exploits a recursion with parameter 2/3.

Applications

The reader is referred to [4] and [6] for a long list of applications of large suffix trees.

Open Problems

The recent theoretical and practical achievements mean the idea that "suffix trees are not practical except when the text size to handle is so small that the suffix tree fits in internal memory" is no longer the case [13]. Given a suffix tree, it is known now (see e. g. [4,10]) how to map it onto a disk-memory system in order to allow I/O-efficient traversals for subsequent pattern searches. A fortiori, suffix-tree storage and construction are challenging problems that need further investigation.

Space optimization is closely related to time optimization in a disk-memory system, so the design of *succinct* suffix-tree implementations is a key issue in order to scale to Gigabytes of data in reasonable time. This topic is an active area of theoretical research with many fascinating solutions (see e. g. [14]), which have not yet been fully explored in the practical setting.

It is theoretically challenging to design a suffix-tree construction algorithm that takes optimal I/Os and space proportional to the *entropy* of the indexed string. The more compressible is the string, the lighter should be the space requirement of this algorithm. Some results are known [7,10,11], but both issues of compression and I/Os have not yet been tackled jointly.

Experimental Results

The interest in building large suffix trees arose in the last few years because of the recent advances in sequencing technology, which have allowed the rapid accumulation of DNA and protein data. Some recent papers [1,2,8,15] proposed new practical algorithms that allow us to scale to Gbps/hours. Surprisingly enough, these algorithms are based on *disk-inefficient* schemes, but they properly select the insertion order of the suffixes and exploit carefully the internal memory as a buffer, so that their performance does not suffers significantly from the theoretical I/O-bottleneck.

In [8] the authors propose an *incremental* algorithm, called $\texttt{PrePar}$, which performs multiple passes over the string S and constructs the suffix tree for a *subrange* of suffixes at each pass. For a user-defined a parameter q, a suffix subrange is defined as the set of suffixes prefixed by the same q-long string. Suffix subranges induce subtrees of $\mathcal{T}_S$ which can thus be built independently, and evicted from internal memory as they are completed. The experiments reported in [8] successfully index 286 Mbps using 2 Gb internal memory.

In [2] the authors propose an improved version of $\texttt{PrePar}$, called $\texttt{DynaCluster}$, that deploys a *dynamic* technique to identify suffix subranges. Unlike `Prepar`, `DynaCluster` does not scan over and over the string S, but it starts from the q-based subranges and then splits them recursively in a DFS-manner if their size is larger than a fixed threshold τ. Splitting is implemented by looking at the next q characters of the suffixes in the subrange. This clustering and lazy-DFS visit of $\mathcal{T}_S$ significantly reduce the number of I/Os incurred by the frequent edge-splitting operations that occur during the suffix tree construction process; and allow it to cope efficiently with skew data. As a result, `DynaCluster` constructs suffix trees for 200Mbps with only 16 Mb internal memory.

More recently, [15] improved the space requirement and the buffering efficiency, thus being able to construct a suffix tree of 3 Gbps in 30 hours; whereas [1] improved the I/O behavior of RAM-algorithms for online suffix-tree construction, by devising a novel low-overhead buffering policy.

Cross References

- ► Cache-Oblivious Sorting
- ► String Sorting
- ► Suffix Array Construction
- ► Suffix Tree Construction in RAM
- ► Text Indexing

Recommended Reading

1. Bedathur, S.J., Haritsa, J.R.: Engineering a fast online persistent suffix tree construction., In: Proc. 20th International Conference on Data Engineering, pp. 720–731, Boston, USA (2004)
2. Cheung, C., Yu, J., Lu, H.: Constructing suffix tree for gigabyte sequences with megabyte memory. IEEE Trans. Knowl. Data Eng. **17**, 90–105 (2005)
3. Farach-Colton, M., Ferragina, P., Muthukrishnan, S.: On the sorting-complexity of suffix tree construction. J. ACM **47** 987–1011 (2000)
4. Ferragina, P.: Handbook of Computational Molecular Biology. In: Computer and Information Science Series, ch. 35 on "String search in external memory: algorithms and data structures". Chapman & Hall/CRC, Florida (2005)
5. Ferragina, P., Grossi, R.: The string B-tree: A new data structure for string search in external memory and its applications. J. ACM **46**, 236–280 (1999)
6. Gusfield, D.: Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology. Cambridge University Press, Cambridge (1997)
7. Hon, W., Sadakane, K., Sung, W.: Breaking a time-and-space barrier in constructing full-text indices. In: IEEE Symposium on Foundations of Computer Science (FOCS), 2003, pp. 251–260
8. Hunt, E., Atkinson, M., Irving, R.: Database indexing for large DNA and protein sequence collections. Int. J. Very Large Data Bases **11**, 256–271 (2002)
9. Kärkkäinen, J., Sanders, P., Burkhardt, S.: Linear work suffix array construction. J. ACM **53**, 918–936 (2006)
10. Ko, P., Aluru, S.: Optimal self-adjusting trees for dynamic string data in secondary storage. In: Symposium on String Processing and Information Retrieval (SPIRE). LNCS, vol. 4726, pp. 184-194. Springer, Berlin (2007)
11. Mäkinen, V., Navarro, G.: Dynamic Entropy-Compressed Sequences and Full-Text Indexes. In: Proc. 17th Symposium on Combinatorial Pattern Matching (CPM). LNCS, vol. 4009, pp. 307–318. Springer, Berlin (2006)
12. Manber, U., Myers, G.: Suffix arrays: a new method for on-line string searches. SIAM J. Comput. **22**, 935–948 (1993)
13. Navarro, G., Baeza-Yates, R.: A hybrid indexing method for approximate string matching. J. Discret. Algorithms **1**, 21–49 (2000)
14. Navarro, G., Mäkinen, V.: Compressed full text indexes. ACM Comput. Surv. **39**(1) (2007)
15. Tata, S., Hankins, R.A., Patel, J.M.: Practical suffix tree construction. In: Proc. 13th International Conference on Very Large Data Bases (VLDB), pp. 36–47, Toronto, Canada (2004)
16. Vitter, J.: External memory algorithms and data structures: Dealing with MASSIVE DATA. ACM Comput. Surv. **33**, 209–271 (2002)

Suffix Tree Construction in RAM

1997; Farach-Colton

JENS STOYE
Department of Technology,
University of Bielefeld, Bielefeld, Germany

Keywords and Synonyms

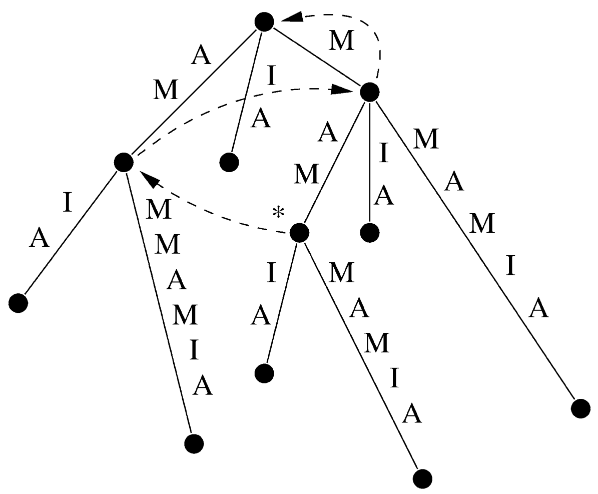
Full-text index construction

Problem Definition

The suffix tree is perhaps the best-known and most-studied data structure for string indexing with applications in many fields of sequence analysis. After its invention in the early 1970s, several approaches for the efficient construction of the suffix tree of a string have been developed for various models of computation. The most prominent of those that construct the suffix tree in main memory are summarized in this entry.

Notations

Given an alphabet Σ, a *trie* over Σ is a rooted tree whose edges are labeled with strings over Σ such that no two labels of edges leaving the same vertex start with the same symbol. A trie is *compacted* if all its internal vertices, except possibly the root, are branching. Given a finite string $S \in \Sigma^n$, the *suffix tree* of S, $T(S)$, is the compacted trie over Σ such that the concatenations of the edge labels along the paths from the root to the leaves are the suffixes of S. An example is given in Fig. 1.



Suffix Tree Construction in RAM, Figure 1
The suffix tree for the string $S = MAMMAMIA$. Dashed arrows denote suffix links that are employed by all efficient suffix tree construction algorithms

The concatenation of the edge labels from the root to a vertex v of $T(S)$ is called the *path-label* of v, $P(v)$. For example, the path-label of the vertex indicated by the asterisk in Fig. 1 is $P(*) = MAM$.

Constraints

The time complexity of constructing the suffix tree of a string S of length n depends on the size of the underlying alphabet Σ. It may be constant, it may be the alphabet of integers $\Sigma = \{1, 2, \ldots, n\}$, or it may be an arbitrary finite set whose elements can be compared in constant time. Note that the latter case reduces to the previous one if one maps the symbols of the alphabet to the set $\{1, \ldots, n\}$, though at the additional cost of sorting Σ.

Problem 1 (suffix tree construction)
INPUT: *A finite string S of length n over an alphabet Σ.*
OUTPUT: *The suffix tree $T(S)$.*

If one assumes that the outgoing edges at each vertex are lexicographically sorted, which is usually the case, the suffix tree allows to retrieve the sorted order of S's characters in linear time. Therefore, suffix tree construction inherits the lower bounds from the problem complexity of sorting: $\Omega(n \log n)$ in the general alphabet case, and $\Omega(n)$ for integer alphabets.

Key Results

Theorem 1 *The suffix tree of a string of length n requires $\Theta(n \log n)$ bits of space.*

This is easy to see since the number of leaves of $T(S)$ is at most n, and so is the number of internal vertices that, by definition, are all branching, as well as the number of edges. In order to see that each edge label can be stored in $O(\log n)$ bits of space, note that an edge label is always a substring of S. Hence it can be represented by a pair (ℓ, r) consisting of *left pointer* ℓ and *right pointer* r, if the label is $S[\ell, r]$.

Note that this space bound is not optimal since there are $|\Sigma|^n$ different strings and hence suffix trees, while $n \log n$ bits would allow to represent $n!$ different entities.

Theorem 2 *Suffix trees can be constructed in optimal time, in particular:*

1. *For constant-size alphabet, the suffix tree $T(S)$ of a string S of length n can be constructed in $O(n)$ time* [11, 12,13]. *For general alphabet, these algorithms require $O(n \log n)$ time.*
2. *For integer alphabet, the suffix tree of S can be constructed in $O(n)$ time* [4,9].

Generally, there is a natural strategy to construct a suffix tree: Iteratively all suffixes are inserted into an initially empty structure. Such a strategy will immediately lead to a linear-time construction algorithm if each suffix can be inserted in constant time. Finding the correct position where to insert a suffix, however, is the main difficulty of suffix tree construction.

The first solution for this problem was given by Weiner in his seminal 1973 paper [13]. His algorithm inserts the suffixes from shortest to longest, and the insertion point is found in amortized constant time for constant-size alphabet, using rather a complicated amount of additional data structures. A simplified version of the algorithm was presented by Chen and Seiferas [3]. They give a cleaner presentation of the three types of links that are required in order to find the insertion points of suffixes efficiently, and their complexity proof is easier to follow. Since the suffix tree is constructed while reading the text from right to left, these two algorithms are sometimes called *anti-online* constructions.

A different algorithm was given 1976 by McCreight [11]. In this algorithm the suffixes are inserted into the growing tree from longest to shortest. This simplifies the update procedure, and the additional data structure is limited to just one type of link: an internal vertex v with path label $P(v) = aw$ for some symbol $a \in \Sigma$ and string $w \in \Sigma^*$ has a *suffix link* to the vertex u with path label $P(u) = w$. In Fig. 1, suffix links are shown as dashed arrows. They often connect vertices above the insertion points of consecutively inserted suffixes, like the vertex with path-label "M" and the root, when inserting suffixes

"MAMIA" and "AMIA" in the example of Fig. 1. This property allows to reach the next insertion point without having to search for it from the root of the tree, thus ensuring amortized constant time per suffix insertion. Note that since McCreight's algorithm treats the suffixes from longest to shortest and the intermediate structures are not suffix trees, the algorithm is not an online algorithm.

Another linear-time algorithm for constant size alphabet is the online construction by Ukkonen [12]. It reads the text from left to right and updates the suffix tree in amortized constant time per added symbol. Again, the algorithm uses suffix links in order to quickly find the insertion points for the suffixes to be inserted. Moreover, since during a single update the edge labels of all leaf-edges need to be extended by the new symbol, it requires a trick to extend all these labels in constant time: all the right pointers of the leaf edges refer to the same *end of string* value, which is just incremented.

An even stronger concept than online construction is *real-time* construction, where the worst-case (instead of amortized) time per symbol is considered. Amir et al. [1] present for general alphabet a suffix tree construction algorithm that requires $O(\log n)$ worst-case update time per every single input symbol when the text is read from right to left, and thus requires overall $O(n \log n)$ time, like the other algorithms for general alphabet mentioned so far. They achieve this goal using a binary search tree on the suffixes of the text, enhanced by additional pointers representing the lexicographic and the textual order of the suffixes, called *Balanced Indexing Structure*. This tree can be constructed in $O(\log n)$ worst-case time per added symbol and allows to maintain the suffix tree in the same time bound.

The first linear-time suffix tree construction algorithm for integer alphabets was given by Farach–Colton [4]. It uses the so-called *odd-even technique* that proceeds in three steps:

1. Recursively compute the compacted trie of all suffixes of S beginning at odd positions, called the *odd tree* T_o.
2. From T_o compute the *even tree* T_e, the compacted trie of the suffixes beginning at even positions in S.
3. Merge T_o and T_e into the whole suffix tree $T(S)$.

The basic idea of the first step is to encode pairs of characters as single characters. Since at most $n/2$ different such characters can occur, these can be radix-sorted and range-reduced to an alphabet of size $n/2$. Thus, the string S of length n over the integer alphabet $\Sigma = \{1, \ldots, n\}$ is translated in $O(n)$ time into a string S' of length $n/2$ over the integer alphabet $\Sigma' = \{1, \ldots, n/2\}$. Applying the algorithm recursively to this string yields the suffix tree of S'. After translating the edge labels from substrings of S' back to substrings of S, some vertices may exist with outgoing edges whose labels start with the same symbol, because two distinct symbols from Σ' may be pairs with the same first symbol from Σ. In such cases, by local modifications of edge labels or adding additional vertices the trie property can be regained and the desired tree T_o is obtained.

In the second step, the odd tree T_o from the first step is used to generate the lexicographically sorted list (*lex-ordering* for short) of the suffixes starting at odd positions. Radix-sorting these with the characters at the preceding even positions as keys yields a lex-ordering of the even suffixes in linear time. Together with the longest common prefixes of consecutive positions that can be computed in linear time from T_o using constant-time lowest common ancestor queries and the identity

$$lcp(l_{2i}, l_{2j}) = \begin{cases} lcp(l_{2i+1}, l_{2j+1}) + 1 & \text{if } S[2i] = S[2j] \\ 0 & \text{otherwise} \end{cases}$$

this ordering allows to reconstruct the even tree T_e in linear time.

In the third step, the two tries T_o and T_e are merged into the suffix tree $T(S)$. Conceptually, this is a straightforward procedure: the two tries are traversed in parallel, and every part that is present in one or both of the two trees, is inserted in the common structure. However, this procedure is simple only if edges are traversed character by character such that common and differing parts can be observed directly. Such a traversal would, however, require $O(n^2)$ time in the worst case, impeding the desired overall linear running time. Therefore, Farach-Colton suggests to use an oracle that tells, for an edge of T_o and an edge of T_e the length of their common prefix. However, the suggested oracle may overestimate this length, and that is why sometimes the tree generated must be corrected, called *unmerging*. The full details of the oracle and the unmerging procedure can be found in [4].

Overall, if $T(n)$ is the time it takes to build the suffix tree of a string $S \in \{1, \ldots, n\}^n$, the first step takes $T(n/2) + O(n)$ time and the second and third step take $O(n)$ time, thus the whole procedure takes $O(n)$ overall time on the RAM model.

Another linear-time construction of suffix trees for integer alphabets can be achieved via linear-time construction of suffix arrays together with longest common prefix tabulation, as described by Kärkkäinen and Sanders in [9].

In some applications the so-called *generalized* suffix tree of several strings is used, a dictionary obtained by constructing the suffix tree of the concatenation of the contained strings. An important question that arises in this context is that of dynamically updating the tree upon insertion and deletion of strings from the dictionary. More specifically, since edge-labels are stored as pairs of pointers into the original string, when deleting a string from the dictionary the corresponding pointers may become invalid and need to be updated. An algorithm to solve this problem in amortized linear time was given by Fiala and Greene [6], a linear worst-case (and hence real-time) algorithm was given by Ferragina et al. [5].

Applications

The suffix tree supports many applications, most of them in optimal time and space, including exact string matching, set matching, longest common substring of two or more sequences, all-pairs suffix-prefix matching, repeat finding, and text compression. These and several other applications, many of them from bioinformatics, are given in [2] and [8].

Open Problems

Some theoretical questions regarding the expected size and branching structure of suffix trees under more complicated than i. i. d. sequence models are still open. Currently most of the research has moved towards more space-efficient data structures like suffix arrays and compressed string indices.

Experimental Results

Suffix trees are infamous for their high memory requirements. The practical space consumption is between 9 and 11 times the size of the string to be indexed, even in the most space-efficient implementations known [7,10]. Moreover, [7] also shows that suboptimal algorithms like the very simple quadratic-time *write-only top-down* (WOTD) algorithm can outperform optimal algorithms on many real-world instances in practice, if carefully engineered.

URL to Code

Several sequence analysis libraries contain code for suffix tree construction. For example, Strmat (http://www.cs.ucdavis.edu/~gusfield/strmat.html) by Gusfield et al. contains implementations of Weiner's and Ukkonen's algorithm. An implementation of the WOTD algorithm by Kurtz can be found at (http://bibiserv.techfak.uni-bielefeld.de/wotd).

Cross References

► Compressed Text Indexing
► String Sorting
► Suffix Array Construction
► Suffix Tree Construction in Hierarchical Memory
► Text Indexing

Recommended Reading

1. Amir, A., Kopelowitz, T., Lewenstein, M., Lewenstein, N.: Towards real-time suffix tree construction. In: Proceedings of the 12th International Symposium on String Processing and Information Retrieval, SPIRE 2005. LNCS, vol. 3772, pp. 67–78. Springer, Berlin (2005)
2. Apostolico, A.: The myriad virtues of subword trees. In: Apostolico, A., Galil, Z. (eds.) Combinatorial Algorithms on Words. NATO ASI Series, vol. F12, pp. 85–96. Springer, Berlin (1985)
3. Chen, M.T., Seiferas, J.: Efficient and elegant subword tree construction. In: Apostolico, A., Galil, Z. (eds.) Combinatorial Algorithms on Words. Springer, New York (1985)
4. Farach, M.: Optimal suffix tree construction with large alphabets. In: Proc. 38th Annu. Symp. Found. Comput. Sci., FOCS 1997, pp. 137–143. IEEE Press, New York (1997)
5. Ferragina, P., Grossi, R., Montangero, M.: A note on updating suffix tree labels. Theor. Comput. Sci. **201**, 249–262 (1998)
6. Fiala, E.R., Greene, D.H.: Data compression with finite windows. Commun. ACM **32**, 490–505 (1989)
7. Giegerich, R., Kurtz, S., Stoye, J.: Efficient implementation of lazy suffix trees. Softw. Pract. Exp. **33**, 1035–1049 (2003)
8. Gusfield, D.: Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology. Cambridge University Press, New York (1997)
9. Kärkkäinen, J., Sanders, P.: Simple linear work suffix array construction. In: Proceedings of the 30th International Colloquium on Automata, Languages, and Programming, ICALP 2003. LNCS, vol. 2719, pp. 943–955. Springer, Berlin (2003)
10. Kurtz, S.: Reducing the space requirements of suffix trees. Softw. Pract. Exp. **29**, 1149–1171 (1999)
11. McCreight, E.M.: A space-economical suffix tree construction algorithm. J. ACM **23**, 262–272 (1976)
12. Ukkonen, E.: On-line construction of suffix trees. Algorithmica **14**, 249–260 (1995)
13. Weiner, P.: Linear pattern matching algorithms. In: Proc. of the 14th Annual IEEE Symposium on Switching and Automata Theory, pp. 1–11. IEEE Press, New York (1973)

Support Vector Machines

1992; Boser, Guyon, Vapnik

NELLO CRISTIANINI[1], ELISA RICCI[2]
[1] Department of Engineering Mathematics, and Computer Science, University of Bristol, Bristol, UK
[2] Department of Engineering Mathematics, University of Perugia, Perugia, Italy

Problem Definition

In 1992 Vapnik and coworkers [1] proposed a supervised algorithm for classification that has since evolved into what are now known as Support Vector Machines

(SVMs) [2]: a class of algorithms for classification, regression and other applications that represent the current state of the art in the field. Among the key innovations of this method were the explicit use of convex optimization, statistical learning theory, and kernel functions.

Classification

Given a *training set* $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_\ell, y_\ell)\}$ of data points $\mathbf{x}_i$ from $X \subseteq \mathbb{R}^n$ with corresponding labels y_i from $Y = \{-1, +1\}$, generated from an unknown distribution, the task of classification is to learn a function $g : X \rightarrow Y$ that correctly classifies new examples $(\mathbf{x}, y)$ (i. e. such that $g(\mathbf{x}) = y$) generated from the same underlying distribution as the training data.

A good classifier should guarantee the best possible generalization performance (e. g. the smallest error on unseen examples). Statistical learning theory [3], from which SVMs originated, provides a link between the expected generalization error for a given training set and a property of the classifier known as its capacity. The SV algorithm effectively regulates the capacity by considering the function corresponding to the hyperplane that separates, according to the labels, the given training data and it is maximally distant from them (*maximal margin hyperplane*). When no linear separation is possible a non-linear mapping into a higher dimensional *feature space* is realized. The hyperplane found in the feature space corresponds to a non-linear decision boundary in the input space.

Let $\phi : I \subseteq \mathbb{R}^n \rightarrow F \subseteq \mathbb{R}^n$ a mapping from the input space I to the feature space F (Fig. 1a). In the learning phase, the algorithm finds a hyperplane defined by the equation $\langle \mathbf{w}, \phi(\mathbf{x}_i) \rangle = b$ such that the *margin*

$$\gamma = \min_{1 \le i \le \ell} y_i(\langle \mathbf{w}, \phi(\mathbf{x}_i)\rangle - b) = \min_{1 \le i \le \ell} y_i g(\mathbf{x}_i) \quad (1)$$

is maximized, where $\langle , \rangle$ denotes the inner product, $\mathbf{w}$ is a ℓ dimensional vector of weights, b is a threshold.

The quantity $(\langle \mathbf{w}, \phi(\mathbf{x}_i)\rangle - b)/\|\mathbf{w}\|$ is the distance of the sample $\mathbf{x}_i$ from the hyperplane. When multiplied by the label y_i it gives a positive value for correct classification and a negative value for an uncorrect one. Given a new data point $\mathbf{x}$ a label is assigned evaluating the decision function:

$$g(\mathbf{x}) = sign(\langle \mathbf{w}, \phi(\mathbf{x})\rangle - b) \quad (2)$$

Maximizing the Margin

For linearly separable classes, there exists a hyperplane $(\mathbf{w}, b)$ such that:

$$y_i(\langle \mathbf{w}, \phi(\mathbf{x}_i)\rangle - b) \ge \gamma \quad i = 1, \ldots, \ell \quad (3)$$

Imposing $\|\mathbf{w}\|^2 = 1$, the choice of the hyperplane such that the margin is maximized is equivalent to the following optimization problem:

$$\begin{aligned} &\max_{\mathbf{w},b,\gamma} \gamma \\ &\text{subject to } y_i(\langle \mathbf{w}, \phi(\mathbf{x}_i)\rangle - b) \ge \gamma \quad i = 1, \ldots, \ell \\ &\qquad \text{and } \|\mathbf{w}\|^2 = 1. \end{aligned} \quad (4)$$

An efficient solution can be found in the dual space by introducing the Lagrange multipliers α_i, $i = 1, \ldots \ell$. The problem (4) can be recast in the following dual form:

$$\begin{aligned} &\max_{\boldsymbol{\alpha}} \sum_{i=1}^{\ell} \alpha_i - \sum_{i=1}^{\ell}\sum_{j=1}^{\ell} \alpha_i \alpha_j y_i y_j \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j)\rangle \\ &\text{subject to } \sum_{i=1}^{\ell} \alpha_i y_i = 0, \quad \alpha_i \ge 0 \end{aligned} \quad (5)$$

This formulation shows how the problem reduces to a *convex* (quadratic) optimization task. A key property of solutions $\boldsymbol{\alpha}^*$ of this kind of problems is that they must satisfy the Karush–Kuhn–Tucker (KKT) conditions, that ensure that only a subset of training examples needs to be associated to a non-zero α_i. This property is called *sparseness* of the SVM solution, and is crucial in practical applications.

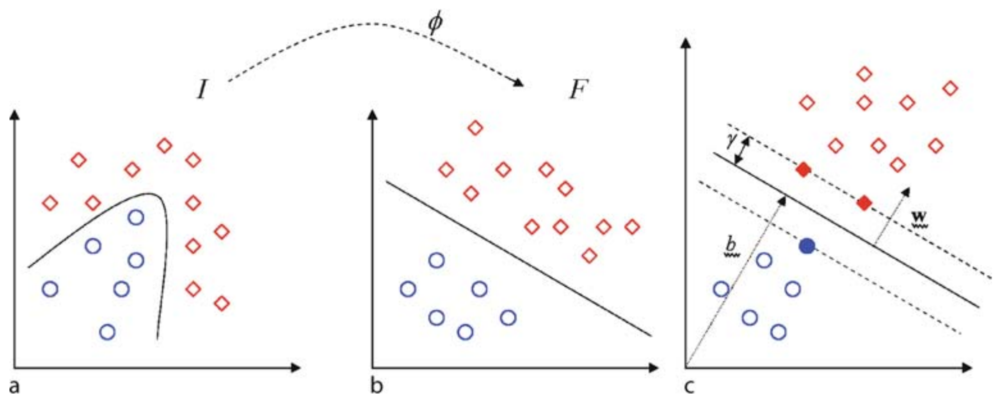
In the solution $\boldsymbol{\alpha}^*$, often only a subset of training examples is associated to non-zero α_i. These are called *support vectors* and correspond to the points that lie closest to the separating hyperplane (Fig. 1b). For the maximal margin hyperplane the weights vector $\mathbf{w}^*$ is given by linear function of the training points:

$$\mathbf{w}^* = \sum_{i=1}^{\ell} \alpha_i^* y_i \phi(\mathbf{x}_i) \quad (6)$$

Then the decision function (2) can equivalently be expressed as:

$$g(\mathbf{x}) = sign(\sum_{i=1}^{\ell} \alpha_i^* y_i \langle \phi(\mathbf{x}_i), \phi(\mathbf{x})\rangle - b) \quad (7)$$

For a support vector $\mathbf{x}_i$, it is $\langle \mathbf{w}^*, \phi(\mathbf{x}_i)\rangle - b = y_i$ from which the optimum bias b^* can be computed. However, it is better to average the values obtained by considering all the support vectors [2]. Both the quadratic programming (QP) problem (5) and the decision function (7) depend only on the dot product between data points. The matrix of dot products with elements $K_{ij} = K(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j)\rangle$ is called the *kernel matrix*. In the case of linear separation $K(\mathbf{x}_i, \mathbf{x}_j) = \langle \mathbf{x}_i, \mathbf{x}_j \rangle$,



Support Vector Machines, Figure 1
a The feature map simplifies the classification task. b A maximal margin hyperplane with its support vectors highlighted

but in general, one can use functions that provide non-linear decision boundaries. Widely used kernels are the polynomial $K(\mathbf{x}_i, \mathbf{x}_j) = (\langle \mathbf{x}_i, \mathbf{x}_j \rangle + 1)^d$ or the Gaussian $K(\mathbf{x}_i, \mathbf{x}_j) = e^{-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{\sigma^2}}$ where d and σ are user-defined parameters.

Key Results

In the framework of learning from examples, SVMs have shown several advantages compared to traditional neural network models (which represented the state of the art in many classification tasks up to 1992). The statistical motivation for seeking the maximal margin solution is to minimize an upper bound on the test error that is independent of the number of dimensions and inversely proportional to the separation margin (and the sample size). This directly suggests embedding of the data in a high-dimensional space where a large separation margin can be achieved; that this can be done efficiently with kernels, and in a convex fashion, are two crucial computational considerations. The sparseness of the solution, implied by the KKT conditions, adds to the efficiency of the result.

The initial formulation of SVMs by Vapnik and coworkers [1] has been extended by many other researchers. Here we summarize some key contributions.

Soft Margin

In the presence of noise the SV algorithm can be subjected to overfitting. In this case one needs to tolerate some training errors in order to obtain a better generalization power. This has led to the development of the *soft margin* classifiers [4]. Introducing the slack variables $\xi_i \geq 0$, optimal class separation can be obtained by:

$$\begin{aligned} &\min_{\mathbf{w},b,\gamma,\boldsymbol{\xi}} \; -\gamma + C\sum_{i=1}^{\ell} \xi_i \\ &\text{subject to } y_i(\langle \mathbf{w}, \phi(\mathbf{x}_i)\rangle - b) \geq \gamma - \xi_i, \; \xi_i \geq 0 \\ &\qquad i = 1, \dots, \ell \;\text{ and }\; \|\mathbf{w}\|^2 = 1. \end{aligned} \tag{8}$$

The constant C is user-defined and controls the trade-off between the maximization of the margin and the number of classification errors. The dual formulation is the same as (5) with the only difference in the bound constraints $(0 \leq \alpha_i \leq C, \quad i = 1, \dots, \ell)$. The choice of soft margin parameter is one of the two main design choices (together with the kernel function) in applications. It is an elegant result [5] that the entire set of solutions for all possible values of C can be found with essentially the same computational cost over finding a single solution: this set is often called the *regularization path*.

Regression

A SV algorithm for regression, called support vector regression (SVR), was proposed in 1996 [6]. A linear algorithm is used in the kernel-induced feature space to construct a function such that the training points are inside a tube of given radius ε. As for classification the regression function only depends on a subset of the training data.

Speeding up the Quadratic Program

Since the emergence of SVMs, many researchers have developed techniques to effectively solve the problem (5): a quite time-consuming task, especially for large training sets. Most methods decompose large-scale problems into a series of smaller ones. The most widely used method is that of Platt [7] and it is known as Sequential Minimal Optimization.

Kernel Methods

In SVMs, both the learning problem and the decision function can be formulated only in terms of dot products between data points. Other popular methods (i. e. Principal Component Analysis, Canonical Correlation Analysis, Fisher Discriminant) have the same property. This fact has led to a huge number of algorithms that effectively use kernels to deal with non-linear functions keeping the same complexity of the linear case. They are referred to as *kernel methods* [8,9].

Choosing the Kernel

The main design choice when using SVMs is the selection of an appropriate kernel function, a problem of model selection that roughly relates to the choice of a topology for a neural network. It is a non-trivial result [10] that also this key task can be translated into a convex optimization problem (a semi-definite program) under general conditions. A kernel can be optimally selected from a kernel space resulting from all linear combinations of a basic set of kernels.

Kernels for General Data

Kernels are not just useful tools to allow us to deploy methods of linear statistics in a non-linear setting. They also allow us to apply them to non-vectorial data: kernels have been designed to operate on sequences, graphs, text, images, and many other kinds of data [8].

Applications

Since their emergence, SVMs have been widely used in a huge variety of applications. To give some examples good results have been obtained in text categorization, hand-written character recognition, and biosequence analysis.

Text Categorization

Automatic text categorization is where text documents are classified into a fixed number of predefined categories based on their content. In the works performed by Joachims [11] and by Dumais et al. [12], documents are represented by vectors with the so-called bag-of-words approach used in the information retrieval field. The distance between two documents is given by the inner product between the corresponding vectors. Experiments on the collection of Reuters news stories showed good results of SVMs compared to other classification methods.

Hand-Written Character Recognition

This is the first real-world task on which SVMs were tested. In particular two publicly available data sets (USPS and NIST) have been considered since they are usually used for benchmarking classifiers. A lot of experiments, mainly summarized in [13], were performed which showed that SVMs can perform as well as other complex systems without incorporating any detailed prior knowledge about the task.

Bioinformatics

SVMs have been widely used also in bioinformatics. For example, Jaakkola and Haussler [14] applied SVMs to the problem of protein homology detection, i. e. the task of relating new protein sequences to proteins whose properties are already known. Brown et al. [15] describe a successful use of SVMs for the automatic categorization of gene expression data from DNA microarrays.

URL to Code

Many free software implementations of SVMs are available at the website

- www.support-vector.net/software.html

Two in particular deserve a special mention for their efficiency:

- *SVMlight*: Joachims T. Making large-scale SVM learning practical. In: Schölkopf B, Burges CJC, and Smola AJ (eds) Advances in Kernel Methods Support Vector Learning, MIT Press, 1999. Software available at http://svmlight.joachims.org
- *LIBSVM*: Chang CC, and Lin CJ, LIBSVM: a library for support vector machines, 2001. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm

Cross References

► PAC Learning
► Perceptron Algorithm

Recommended Reading

1. Boser, B., Guyon, I., Vapnik, V.: A training algorithm for optimal margin classifiers. In: Proceedings of the Fifth Annual Workshop on Computational Learning Theory, Pittsburgh (1992)
2. Cristianini, N., Shawe-Taylor, J.: An Introduction to Support Vector Machines and other kernel-based learning methods. Cambridge University Press, Cambrigde, Book website: www.support-vector.net (2000)

3. Vapnik, V.: The Nature of Statistical Learning Theory. Springer, New York (1995)
4. Cortes, C., Vapnik, V.: Support-vector network. Mach. Learn. **20**, 273–297 (1995)
5. Hastie, T., Rosset, S., Tibshirani, R., Zhu, J.: The entire regularization path for the support vector machine. J. Mach. Learn. Res. **5**, 1391–1415 (2004)
6. Drucker, H., Burges, C.J.C., Kaufman, L., Smola, A., Vapnik, V.: Support Vector Regression Machines. Adv. Neural. Inf. Process. Syst. (NIPS) **9**, 155–161 MIT Press (1997)
7. Platt, J.: Fast training of support vector machines using sequential minimal optimization. In: Schölkopf, B., Burges, C.J.C., Smola, A.J. (eds.) Advances in Kernel Methods Support Vector Learning. pp 185–208. MIT Press, Cambridge (1999)
8. Shawe-Taylor, J., Cristianini, N.: Kernel Methods for Pattern Analysis. Cambridge University Press, Cambridge. Book website: www.kernel-methods.net (2004)
9. Scholkopf, B., Smola, A.J.: Learning with Kernels. MIT Press, Cambridge (2002)
10. Lanckriet, G.R.G., Cristianini, N., Bartlett, P., El Ghaoui, L., Jordan, M.I.: Learning the Kernel Matrix with Semidefinite Programming. J. Mach. Learn. Res. **5**, 27–72 (2004)
11. Joachims, T.: Text categorization with support vector machines. In: Proceedings of European Conference on Machine Learning (ECML) Chemnitz (1998)
12. Dumais, S., Platt, J., Heckerman, D., Sahami, M.: Inductive learning algorithms and representations for text categorization. In: 7th International Conference on Information and Knowledge Management (1998)
13. LeCun, Y., Jackel, L.D., Bottou, L., Brunot, A., Cortes, C., Denker, J.S., Drucker, H., Guyon, I., Muller, U.A., Sackinger, E., Simard, P., Vapnik, V.: Comparison of learning algorithms for handwritten digit recognition. In: Fogelman-Soulie F., Gallinari P. (eds.), Proceedings International Conference on Artificial Neural Networks (ICANN) **2**, 5360. EC2 (1995)
14. Jaakkola, T.S., Haussler, D.: Probabilistic kernel regression models. In: Proceedings of the 1999 Conference on AI and Statistics Fort Lauderdale (1999)
15. Brown, M., Grundy, W., Lin, D., Cristianini, N., Sugnet, C., Furey, T., Ares Jr., M., Haussler, D.: Knowledge-based analysis of microarray gene expression data using support vector machines. In: Proceedings of the National Academy of Sciences **97**(1), 262–267 (2000)

Symbolic Model Checking

1990; Burch, Clarke, McMillan, Dill

Amit Prakash[1], Adnan Aziz[2]
[1] Microsoft, MSN, Redmond, WA, USA
[2] Department of Electrical and Computer Engineering, University of Texas, Austin, TX, USA

Keywords and Synonyms

Formal hardware verification

Problem Definition

Design verification is the process of taking a design and checking that it works correctly. More specifically, every design verification paradigm has three components [6]—(1.) a language for specifying the design in an unambiguous way, (2.) a language for specifying properties that are to be checked of the design, and (3.) a checking procedure, which determines whether the properties hold of the design.

The verification problem is very general: it arises in low-level designs, e. g., checking that a combinational circuit correctly implements arithmetic, as well as high-level designs, e. g., checking that a library written in high-level language correctly implements an abstract data type.

Hardware Verification

The verification of hardware designs is particularly challenging. Verification is difficult in part because the large number of concurrent operations, make it very difficult to conceive of and construct all possible corner-cases, e. g., one unit initiating a transaction at the same cycle as another receiving an exception. In addition, software models used for simulation run orders of several magnitude slower than the final chip operates at. Faulty hardware is usually impossible to correct after fabrication, which means that the cost of a defect is very high, since it takes several months to go through the process of designing and fabricating new hardware. Wile et al. [15] provide a comprehensive account of hardware verification.

State Explosion

Since the number of state holding elements in digital hardware is bounded, the number of possible states that the design can be in is infinite, so complete automated verification is, in principle, possible. However, the number of states that a hardware design can reach from the initial state can be exponential in the size of the design; this phenomenon is referred to as "state explosion." In particular, algorithms for verifying hardware that explicitly record visited states, e. g., in a hash table, have very high time complexity, making them infeasible for all but the smallest designs. The problem of complete hardware verification is known to be PSPACE-hard, which means that any approach must be based on heuristics.

Hardware Model

A hardware design is formally described using *circuits* [4,8]. A *combinational circuit* consists of *Boolean combinational elements* connected by *wires*. The Boolean combinational elements are *gates* and *primary inputs*. Gates come in three types: *NOT*, *AND*, and *OR*. The NOT gate functions as follows: it takes a single Boolean-valued *input*, and produces a single Boolean-valued *output* which takes value 0 if the input is 1, and 1 if the input is 0. The AND gate takes two Boolean-valued inputs and produce a single output; the output is 1 if both inputs are 1, and 0 otherwise. The OR gate is similar to AND, except that its output is 1 if one or both inputs are 1. A circuit can be represented as a directed graph where the nodes represent the gates and wires represent edges in the direction of signal flow.

A circuit can be represented by a directed graph where the nodes represent the gates and primary inputs, and edges represent wires in the direction of signal flow. Circuits are required to be acyclic, that is there is no cycle of gates. The absence of cycles implies that a Boolean-assignment to the primary inputs can be propagated through the gates in topological order.

A *sequential circuit* extends the notion of circuit described above by adding *stateful elements*. Specifically, a sequential circuit includes *registers*. Each register has a single input, which is referred to as its *next-state input*.

A *valuation* on a set V is a function whose domain is V. A *state* in a sequential circuit is a Boolean-valued valuation on the set of registers. An *input* to a sequential circuit is a Boolean-valued valuation on the set of primary inputs. Given a state s and an input i, the logic gates in the circuit uniquely define a Boolean-valued valuation t to the set of register inputs—this is referred to as the next state of the circuit at state s under input i, and say s *transitions* to t on input i. It is convenient to denote such a transition by $s \xrightarrow{i} t$.

A sequential circuit can naturally be identified with a *finite state machine* (FSM), which is a graph defined over the set of all states; an edge (s, t) exists in the FSM graph if there exists an input i, state s transitions to t on input i.

Invariant Checking

An *invariant* is a set of states; informally, the term is used to refer to a set of states that are "good" in some sense. One common way to specify an invariant is to write a Boolean formula on the register variables—the states which satisfy the formula are precisely the states in the invariant.

Given states r and s, define r to be *reachable* from s if there is a sequence of inputs $\langle i_0, i_1, \ldots, i_{n-1} \rangle$ such that $s = s_0 \xrightarrow{i_0} s_1 \xrightarrow{i_1} \cdots s_n = t$. A fundamental problem in hardware verification is the following—given an invariant A, and a state s, does there exists a state r reachable from s which is not in A?

Key Results

Symbolic model checking (SMC) is a heuristic approach to hardware verification. It is based on the idea that rather than representing and manipulating states one-at-a-time, it is more efficient to use symbolic expressions to represent and manipulate sets of states.

A key idea in SMC is that given a set $A \subset \{0,1\}^n$, a Boolean function A can be constructed such that $f_A\colon \{0,1\}^n \mapsto \{0,1\}$ given by $f(\alpha_1, \ldots, \alpha_n) = 1$ iff $(\alpha_1, \ldots, \alpha_n) \in A$. Note that given a characteristic function f_A, A can be obtained and vice versa.

There are many ways in which a Boolean function can be represented—formulas in DNF, general Boolean formulas, combinational circuits, etc. In addition to an efficient representation for state sets, the ability to perform fast computations with sets of states is also important—for example, in order to determine if an invariant holds, it is required to compute the set of states reachable from a given state. BDDs [2] are particularly well-suited to representing Boolean functions, as they combine succinct representation with efficient manipulation; they are the data structure underlying SMC.

Image Computation

A key computation that arises in verification is determining the *image* of a set of states A in a design D—the image of A is the set of all states t for which there exists a state in A and an input i such that state s transitions to t under input i. The image of A is denoted by Img(A).

The *transition relation* of a design is the set of (s, i, t) triples such that s transitions to t under input i. Let the design have n registers, and m primary inputs; then the transition relation is subset of $\{0,1\}^n \times \{0,1\}^m \times \{0,1\}^n$.

Conceptually, the transition relation completely captures the dynamics of the design—given an initial state, and input sequence, the evolution of the design is completely determined by the transition relation.

Since the transition relation is a subset of $\{0,1\}^{n+m+n}$, it has a characteristic function $f_T\colon \{0,1\}^{n+m+n} \mapsto \{0,1\}$. View f_T as being defined over the variables $x_0, \ldots, x_{n-1}$, $i_0, \ldots, i_{m-1}, y_0, \ldots, y_{n-1}$. Let the set of states A be represented by the function f_A defined over variables $x_0, \ldots, x_{n-1}$. Then the following identity holds

$$\mathrm{Img}(A) = (\exists x_0 \cdot \exists x_{n-1} \exists i_0 \cdots \exists i_{m-1})(f_A \cdot f_T)\,.$$

The identity hold because $(\beta_0, \ldots, \beta_{n-1})$ satisfies the right-hand side expression exactly when there are values $\alpha_0, \ldots, \alpha_{n-1}$ and $\iota_0, \ldots, \iota_{m-1}$ such that $(\alpha_0, \ldots, \alpha_{n-1}) \in A$ and the state $(\alpha_0, \ldots, \alpha_{n-1})$ transitions to $(\beta_0, \ldots, \beta_{n-1})$ on input $(\iota_0, \ldots, \iota_{m-1})$.

Invariant Checking

The set of all states reachable from a given set A is the limit as n tends to infinity of the sequence of states $\langle R_0, R_1, \dots \rangle$ defined below:

$$R_0 = A$$
$$R_{i+1} = R_i \cup \mathrm{Img}(R_i)\ .$$

Since for all i, $R_i \subseteq R_{i+1}$ and the number of distinct state sets is finite, the limit is reached in some finite number of steps, i. e., for some n, it must be that $R_{n+1} = R_n$. It is straightforward to show that the limit is exactly equal to the set of states reachable from A—the basic idea is to inductively construct input sequences that lead from states in A to R_i, and to show that state t is reachable from a state in A under an input sequence of length l, then t must be in R_l.

Given BDDs F and G representing functions f and g respectively, there is an algorithm based on dynamic programming for performing conjunction, i. e., for computing the BDD for $f \cdot g$. The algorithm has polynomial complexity, specifically $O(|F| \cdot |G|)$, where $|B|$ denotes the number of nodes in the BDD B. There are similar algorithms for performing disjunction $(f + g)$, and computing cofactors (f_x and $f_{x'}$). Together these yield an algorithm for the operation of existential quantification, since $(\exists x) f = f_x + f_{x'}$.

It is straightforward to build BDDs for f_A and f_T: A is typically given using a propositional formula, and the BDD for f_A can be built up using functions for conjunction, disjunction, and negation. The BDD for f_T is built using from the BDDs for the next-state nodes, over the register and primary input variables. Since the only gate types are AND, OR, and NOT, the BDD can be built using the standard BDD operators for conjunction, disjunction, and negation. Let the next state functions be $f_0, \dots, f_{n-1}$; then f_T is $(y_0 = f_0) \cdot (y_1 = f_1) \cdot \dots \cdot (y_{n-1} = f_{n-1})$, and so the BDD for f_T can be constructed using the usual BDD operators.

Since the image computation operation can be expressed in terms of f_A and F_T, and conjunction and existential quantification operations, it can be performed using BDDs. The computation of R_i involves an image operation, and a disjunction, and since BDDs are canonical, the test for fixed-point is trivial.

Applications

The primary application of the technique described above is for checking properties of hardware designs. These properties can be invariants described using propositional formulae over the register variables, in which case the approach above is directly applicable. More generally, properties can be expressed in a *temporal logic* [5], specifically through formulae which express acceptable sequences of outputs and transitions.

CTL is one common temporal logic. A CTL formula is given by the following grammar: if x is a variable corresponding to a register, then $\mathbf{x}$ is a CTL formula; otherwise, if φ and ψ are CTL formulas, then so as $(\neg\phi)$, $(\phi \vee \psi)$, $(\phi \wedge \psi)$, $(\phi \rightarrow \psi)$, and $EX\phi$, $E\phi U\psi$, and $EG\phi$.

A CTL formula is interpreted as being true at a state; a formula $\mathbf{x}$ is true at a state if that register is 1 in that state. Propositional connectives are handled in the standard way, e. g., a state satisfies a formula $(\phi \wedge \psi)$ if it satisfies both φ and ψ. A state s satisfies $EX\phi$ if there exists a state t such that s transitions to, and t satisfies φ. A state s satisfies $E\phi U\psi$ if there exists a sequence of inputs $\langle i_0, \dots, i_n \rangle$ leading through state $\langle s_0 = s, s_1, s_2, \dots, s_{n+1} \rangle$ such that s_{n+1} satisfies ψ, and all states $s_i, i \leq n+1$ satisfy φ. A state s satisfies $EG\phi$ if there exists an infinite sequence of inputs $\langle i_0, i_1, \dots \rangle$ leading through state $\langle s_0 = s, s_1, s_2, \dots \rangle$ such that all states s_i satisfy φ.

CTL formulas can be checked by a straightforward extension of the technique described above for invariant checking. One approach is to compute the set of states in the design satisfying subformulas of φ, starting from the subformulas at the bottom of the parse tree for φ. A minor difference between invariant checking and this approach, is that the latter relies on *pre-image* computation; the pre-image of A is the set of all states t for which there exists an input i such that t transitions under i to a state in A.

Symbolic analysis can also be used to check the equivalence of two designs by forming a new design which operates the two initial designs in parallel, and has a single output that is set to 1 if the two initial designs differ [14]. In practice this approach is too inefficient to be useful, and techniques which rely more on identifying common substructures across designs are more successful.

The complement of the set of reachable states can be used to identify parts of the design which are redundant, and to propagate don't care conditions from the input of the design to internal nodes [12].

Many of the ideas in SMC can be applied to software verification—the basic idea is to "finitize" the problem, e. g., by considering integers to lie in a restricted range, or setting an a priori bound on the size of arrays [7].

Experimental Results

Many enhancements have been made to the basic approach described above. For example, the BDD for the entire transition relation can grow large, so *partitioned tran-*

sition relations [11] are used instead; these are based on the observation that $\exists x.(f \cdot g) = f \cdot \exists x.g$, in the special case that f is independent of x. Another optimization is the use of *don't cares*; for example when computing the image of A, the BDD for f_T can be simplified with respect to transitions originating at A' [13]. Techniques based on SAT have enjoyed great success recently. These approach case the verification problem in terms of satisfiability of a CNF formula. They tend to be used for bounded checks, i. e., determining that a given invariant holds on all input sequences of length k [1]. Approaches based on *transformation-based verification*, complement symbolic model checking by simplifying the design prior to verification. These simplifications typically remove complexity that was added for performance rather than functionality, e. g., pipeline registers.

The original paper by Clarke et al. [3] reported results on a toy example, which could be described in a few dozen lines of a high-level language. Currently, the most sophisticated model checking tool for which published results are ready is SixthSense, developed at IBM [10].

A large number of papers have been published on applying SMC to academic and industrial designs. Many report success on designs with an astronomical number of states—these results become less impressive when taking into consideration the fact that a design with n registers has 2^n states.

It is very difficult to define the complexity of a design. One measure is the number of registers in the design. Realistically, a hundred registers is at the limit of design complexity that can be handles using symbolic model checking. There are cases of designs with many more registers that have been successfully verified with symbolic model checking, but these registers are invariably part of a very regular structure, such as a memory array.

Data Sets

The SMV system described in [9] has been updated, and its latest incarnation nuSMV (http://nusmv.irst.itc.it/) include a number of examples.

The VIS (http://embedded.eecs.berkeley.edu/pubs/downloads/vis) system from UC Berkeley and UC Boulder also includes a large collection of verification problems, ranging from simple hardware circuits, to complex multiprocessor cache systems.

The SIS (http://embedded.eecs.berkeley.edu/pubs/downloads/sis/) system from UC Berkeley is used for logic synthesis. It comes with a number of sequential circuits that have been used for benchmarking symbolic reachability analysis.

Cross References

▶ Binary Decision Graph

Recommended Reading

1. Biere, A., Cimatti, A., Clarke, E., Fujita, M., Zhu, Y.: Symbolic Model Checking Using Sat Procedures Instead of BDDs. In: ACM Design Automation Conference. (1999)
2. Bryant, R.: Graph-based Algorithms for Boolean Function Manipulation. IEEE Trans. Comp. **C-35**, 677–691 (1986)
3. Burch, J.R., Clarke, E.M., McMillan, K.L., Dill, D.L.: Symbolic Model Checking: 10^{20} States and Beyond. Inf. Comp. **98**(2), 142–170 (1992)
4. Cormen, T.H., Leiserson, C.E., Rivest, R.H., Stein, C.: Introduction to Algorithms. MIT Press (2001)
5. Emerson, E.A.: Temporal and Modal Logic. In: van Leeuwen, J. (ed.) Formal Models and Semantics, vol. B of Handbook of Theoretical Computer Science, pp. 996–1072. Elsevier Science (1990)
6. Gupta, A.: Formal Hardware Verification Methods: A Survey. Formal Method Syst. Des. **1**, 151–238 (1993)
7. Jackson, D.: Software Abstractions: Logic, Language, and Analysis. MIT Press (2006)
8. Katz, R.: Contemporary logic design. Benjamin/Cummings Pub. Co. (1993)
9. McMillan, K.L.: Symbolic Model Checking. Kluwer Academic Publishers (1993)
10. Mony, H., Baumgartner, J., Paruthi, V., Kanzelman, R., Kuehlmann, A.: Scalable Automated Verification via Expert-System Guided Transformations. In: Formal Methods in CAD. (2004)
11. Ranjan, R., Aziz, A., Brayton, R., Plessier, B., Pixley, C.: Efficient BDD Algorithms for FSM Synthesis and Verification. In: Proceedings of the International Workshop on Logic Synthesis, May 1995
12. Savoj, H.: Don't Cares in Multi-Level Network Optimization. Ph. D. thesis, University of California, Berkeley, Electronics Research Laboratory, College of Engineering. University of California, Berkeley, CA (1992)
13. Shiple, T.R., Hojati, R., Sangiovanni-Vincentelli, A.L., Brayton, R.K.: Heuristic Minimization of BDDs Using Don't Cares. In: ACM Design Automation Conference, San Diego, CA, June (1994)
14. Touati, H., Savoj, H., Lin, B., Brayton, R.K., Sangiovanni-Vincentelli, A.L.: Implicit State Enumeration of Finite State Machines using BDDs. In: IEEE International Conference on Computer-Aided Design, pp. 130–133, November (1990)
15. Wile, B., Goss, J., Roesner, W.: Comprehensive Functional Verification. Morgan-Kaufmann (2005)

Synchronizers, Spanners

1985; Awerbuch

MICHAEL ELKIN
Department of Computer Science,
Ben-Gurion University, Beer-Sheva, Israel

Keywords and Synonyms

Network synchronization; Low-stretch spanning subgraphs

Problem Definition

Consider a communication network, modeled by an n-vertex undirected unweighted graph $G = (V, E)$, for some positive integer n. Each vertex of G hosts a processor of unlimited computational power; the vertices have unique identity numbers, and they communicate via the edges of G by sending messages of size $O(\log n)$ each.

In the *synchronous* setting the communication occurs in discrete *rounds*, and a message sent in the beginning of a round R arrives at its destination before the round R ends. In the *asynchronous* setting each vertex maintains its own clock, and clocks of distinct vertices may disagree. It is assumed that each message sent (in the asynchronous detting) arrives at its destination within a certain time τ after it was sent, but the value of τ is not known to the processors.

It is generally much easier to devise algorithms that apply to the synchronous setting (henceforth, synchronous algorithms) rather than to the asynchronous one (henceforth, asynchronous algorithms). In [1] Awerbuch initiated the study of simulation techniques that translate synchronous algorithms to asynchronous ones. These simulation techniques are called *synchronizers*.

To devise the first synchronizers Awerbuch [1] constructed a certain graph partition which is of its own interest. In particular, Peleg and Schäffer noticed [8] that this graph partition induces a subgraph with certain interesting properties. They called this subgraph a *graph spanner*. Formally, for an integer positive parameter k, a k-spanner of a graph $G = (V, E)$ is a subgraph $G' = (V, H)$, $H \subseteq E$, such that for every edge $e = (v, u) \in E$, the distance between the vertices v and u in H, $\text{dist}_{G'}(v, u)$, is at most k.

Key Results

Awerbuch devised three basic synchronizers, called α, β, and γ. The synchronizer α is the simplest one; using it results in only a constant overhead in time, but in a very significant overhead in communication. Specifically, the latter overhead is linear in the number of edges of the underlying network. Unlike the synchronizer α, the synchronizer β requires a somewhat costly initialization stage. In addition, using it results in a significant time overhead (linear in the number of vertices n), but it is more communication-efficient than α. Specifically, its communication overhead is linear in n.

Finally, the synchronizer γ represents a tradeoff between the synchronizers α and β. Specifically, this synchronizer is parametrized by a positive integer parameter k. When k is small then the synchronizer behaves similarly to the synchronizer α, and when k is large it behaves similarly to the synchronizer β. A particularly important choice of k is $k = \log n$. At this point on the tradeoff curve the synchronizer γ has a logarithmic in n time overhead, and a linear in n communication overhead. The synchronizer γ has, however, a quite costly initialization stage.

The main result of [1] concerning spanners is that for every $k = 1, 2, \ldots$, and every n-vertex unweighted undirected graph $G = (V, E)$, there exists an $O(k)$-spanner with $O(n^{1+1/k})$ edges. (This result was explicated by Peleg and Schäffer [8].)

Applications

Synchronizers are extensively used for constructing asynchronous algorithms. The first applications of synchronizers are constructing the *breadth-first-search tree* and computing the *maximum flow*. These applications were presented and analyzed by Awerbuch in [1]. Later synchronizers were used for maximum matching [10], for computing shortest paths [7], and for other problems.

Graph spanners were found useful for a variety of applications in distributed computing. In particular, some constructions of synchronizers employ graph spanners [1,9]. In addition, spanners were used for routing [4], and for computing almost shortest paths in graphs [5].

Open Problems

Synchronizers with improved properties were devised by Awerbuch and Peleg [3], and Awerbuch et al. [2]. Both these synchronizers have polylogarithmic time and communication overheads. However, the synchronizers of Awerbuch and Peleg [3] require a large initialization time. (The latter is at least linear in n.) On the other hand, the synchronizers of [2] are randomized. A major open problem is to obtain *deterministic* synchronizers with polylogarithmic time and communication overheads, and sublinear in n initialization time. In addition, the degrees of the logarithm in the polylogarithmic time and communication overheads in synchronizers of [2,3] are quite large. Another important open problem is to construct synchronizers with improved parameters.

In the area of spanners, spanners that distort large distances to a significantly smaller extent than they distort small distances were constructed by Elkin and Peleg in [6]. These spanners fall short from achieving a *purely additive* *distortion*. Constructing spanners with a purely additive distortion is a major open problem.

Cross References

► Sparse Graph Spanners

Recommended Reading

1. Awerbuch, B.: Complexity of network synchronization. J. ACM **4**, 804–823 (1985)
2. Awerbuch, B., Patt-Shamir, B., Peleg, D., Saks, M.E.: Adapting to asynchronous dynamic networks. In: Proc. of the 24th Annual ACM Symp. on Theory of Computing, Victoria, 4–6 May 1992, pp. 557–570
3. Awerbuch, B., Peleg, D.: Network synchronization with polylogarithmic overhead. In: Proc. 31st IEEE Symp. on Foundations of Computer Science, Sankt Louis, 22–24 Oct. 1990, pp. 514–522
4. Awerbuch, B., Peleg, D.: Routing with polynomial communication-space tradeoff. SIAM J. Discret. Math. **5**, 151–162 (1992)
5. Elkin, M.: Computing Almost Shortest Paths. In: Proc. 20th ACM Symp. on Principles of Distributed Computing, Newport, RI, USA, 26–29 Aug. 2001, pp. 53–62
6. Elkin, M., Peleg, D.: Spanner constructions for general graphs. In: Proc. of the 33th ACM Symp. on Theory of Computing, Heraklion, 6–8 Jul. 2001, pp. 173–182
7. Lakshmanan, K.B., Thulasiraman, K., Comeau, M.A.: An efficient distributed protocol for finding shortest paths in networks with negative cycles. IEEE Trans. Softw. Eng. **15**, 639–644 (1989)
8. Peleg, D., Schäffer, A.: Graph spanners. J. Graph Theory **13**, 99–116 (1989)
9. Peleg, D., Ullman, J.D.: An optimal synchronizer for the hypercube. SIAM J. Comput. **18**, 740–747 (1989)
10. Schieber, B., Moran, S.: Slowing sequential algorithms for obtaining fast distributed and parallel algorithms: Maximum matchings. In: Proc. of 5th ACM Symp. on Principles of Distributed Computing, Calgary, 11–13 Aug. 1986, pp. 282–292

T

Table Compression

2003; Buchsbaum, Fowler, Giancarlo

ADAM L. BUCHSBAUM[1], RAFFAELE GIANCARLO[2]
[1] Shannon Laboratory, AT&T Labs, Inc., Florham Park, NJ, USA
[2] Department of Mathematics and Computer Science, University of Palermo, Palermo, Italy

Keywords and Synonyms

Compression of multi-dimensional data; Storage, compression and transmission of tables; Compressive estimates of entropy

Problem Definition

Table compression was introduced by Buchsbaum et al. [2] as a unique application of compression, based on several distinguishing characteristics. Tables are collections of fixed-length records and can grow to be terabytes in size. They are often generated by information systems and kept in data warehouses to facilitate ongoing operations. These data warehouses will typically manage many terabytes of data online, with significant capital and operational costs. In addition, the tables must be transmitted to different parts of an organization, incurring additional costs for transmission. Typical examples are tables of transaction activity, like phone calls and credit card usage, which are stored once but then shipped repeatedly to different parts of an organization: for fraud detection, billing, operations support, etc. The goals of table compression are to be fast, online, and effective: eventual compression ratios of 100:1 or better are desirable. Reductions in required storage and network bandwidth are obvious benefits.

Tables are different than general databases [2]. Tables are written once and read many times, while databases are subject to dynamic updates. Fields in table records are fixed in length, and records tend to be homogeneous; database records often contain intermixed fixed- and variable-length fields. Finally, the goals of compression differ. Database compression stresses index preservation, the ability to retrieve an arbitrary record, under compression [6]. Tables are typically not indexed at the level of individual records; rather, they are scanned in toto by downstream applications.

Consider each record in a table to be a row in a matrix. A naive method of table compression is to compress the string derived from scanning the table in row-major order. Buchsbaum et al. [2] observe experimentally that partitioning the table into contiguous intervals of columns and compressing each interval separately in this fashion can achieve significant compression improvement. The partition is generated by a one-time, offline training procedure, and the resulting compression strategy is applied online to the table. In their application, tables are generated continuously, so offline training time can be ignored. They also observe heuristically that certain rearrangements of the columns prior to partitioning further improve compression by grouping dependent columns more closely. For example, in a table of addresses and phone numbers, the area code can often be predicted by the zip code when both are defined geographically. In information-theoretic terms, these dependencies are *contexts*, which can be used to predict parts of a table. Analogously to strings, where knowledge of context facilitates succinct codings of a symbols, the existence of contexts in tables implies, in principle, the existence of a more succinct representation of the table.

Two main avenues of research have followed, one based on the notion of combinatorial dependency [2,3] and the other on the notion of column dependency [14, 15]. The first formalizes dependencies analogously to the joint entropy of random variables, while the second does so analogously to conditional entropy [7]. These approaches to table compression have deep connections to universal similarity metrics [11], based on Kolmogorov complexity and compression, and their later uses in classification [5]. Both approaches are instances of a new emerging paradigm for data compression, referred to as *boost-*

ing [8], where data are reorganized to improve the performance of a given compressor. A software platform to facilitate the investigation of such invertible data transformations is described by Vo [16].

Notations

Let T be a table of $n = |T|$ columns and m rows. Let $T[i]$ denote the ith column of T. Given two tables T_1 and T_2, let $T_1 T_2$ be the table formed by their juxtaposition. That is, $T = T_1 T_2$ is defined so that $T[i] = T_1[i]$ for $1 \leq i \leq |T_1|$ and $T[i] = T_2[i - |T_1|]$ for $|T_1| < i \leq |T_1| + |T_2|$. We use the shorthand $T[i, j]$ to represent the projection $T[i] \cdots T[j]$ for any $j \geq i$. Also, given a sequence P of column indices, we denote by $T[P]$ the table obtained from T by projecting the columns with indices in P.

Combinatorial Dependency and Joint Entropy of Random Variables

Fix a compressor C: e. g., gzip, based on LZ77 [17]; compress, based on LZ78 [18]; or bzip, based on Burrows–Wheeler [4]. Let $H_C(T)$ be the size of the result of compressing table T as a string in row-major order using C. Let $H_C(T_1, T_2) = H_C(T_1 T_2)$.$H_C(\cdot)$ is thus a cost function defined on the ordered power set of columns. Two tables T_1 and T_2, which might be projections of columns from a common table T, are *combinatorially dependent* if $H_C(T_1, T_2) < H_C(T_1) + H_C(T_2)$ – if compressing them together is better than compressing them separately – and *combinatorially independent* otherwise. Buchsbaum et al. [3] show that combinatorial dependency is a compressive estimate of statistical dependency when formalized by the joint entropy of two random variables, i. e., the statistical relatedness of two objects is measured by the gain realized by compressing them together rather than separately. Indeed, combinatorial dependency becomes statistical dependency when H_C is replaced by the joint entropy function [7]. Analogous notions starting from Kolmogorov complexity are derived by Li et al. [11] and used for classification and clustering [5]. Figure 1 exemplifies why rearranging and partitioning columns may improve compression.

Problem 1 *Find a partition $\mathcal{P}$ of T into sets of contiguous columns that minimizes $\sum_{Y \in \mathcal{P}} H_C(Y)$ over all such partitions.*

Problem 2 *Find a partition $\mathcal{P}$ of T that minimizes $\sum_{Y \in \mathcal{P}} H_C(Y)$ over all partitions.*

The difference between Problems 1 and 2 is that the latter does not require the parts of $\mathcal{P}$ to be sets of contiguous columns.

9	0	8	2	7	3
9	0	8	3	7	5
9	0	8	5	7	6
9	0	8	2	7	5

Table Compression, Figure 1
The first three columns of the table, taken in row-major order, form a repetitive string that can be very easily compressed. Therefore, it may be advantageous to compress these columns separately. If the fifth column is swapped with the fourth, we get an even longer repetitive string that, again, can be compressed separately from the other two columns

Column Dependency and Conditional Entropy of Random Variables

Definition 1 For any table T, a *dependency relation* is a pair (P, c) in which P is a sequence of distinct column indices (possibly empty) and $c \notin P$ is another column index. If the length of P is less than or equal to k, then (P, c) is called a k-relation. P is the *predictor sequence* and c is the *predictee.*

Definition 2 Given a dependency relation (P, c), the *dependency transform* $\mathrm{dt}_P(c)$ of c is formed by permuting column $T[c]$ based on the permutation induced by a stable sort of the rows of P.

Definition 3 A collection D of dependency relations for table T is said to be a k-transform if and only if: (a) each column of T appears exactly once as a predictee in some dependency relation (P, c); (b) the dependency hypergraph $G(D)$ is acyclic; (c) each dependency relation (P, c) is a k-relation.

Let $\omega(P, c)$ be the cost of the dependency relation (P, c), and let $\delta(m)$ be an upper bound on the cost of computing $\omega(P, c)$. Intuitively, $\omega(P, c)$ gives an estimate of how well a rearrangement of column c will compress, using the rows of P as contexts for its symbols. We will provide an example after the formal definitions.

Problem 3 *Find a k-transform D of minimum cost $\omega(D) = \sum_{(P,c) \in D} \omega(P, c)$.*

Definition 1 extends to columns the notion of context that is well known for strings. Definition 3 defines a microtransformation that reorganizes the column symbols by grouping together those that have similar contexts. The context of a column symbol is given by the corresponding row in $T[P]$. The fundamental ideas here are the same as in the Burrows and Wheeler transform [4]. Finally, Problem 3 asks for an optimal strategy to reorganize the data prior to compression. The cost function ω provides an estimate of how well c can be compressed using the knowledge of $T[P]$.

Vo and Vo [14] connect these ideas to the conditional entropy of random variables. Let S be a sequence, $\mathcal{A}(S)$ its distinct elements, and f_a the frequency of each element a. The *zeroth-order empirical entropy* of S [13] is

$$H_0(S) = -\frac{1}{|S|} \sum_{\alpha \in \mathcal{A}(S)} f_a \lg \frac{f_a}{|S|} ,$$

and the *modified zeroth order empirical entropy* [13] is

$$H_0^*(S) = \begin{cases} 0 & \text{if } |S| = 0 , \\ (1 + \lg |S|)/|S| & \text{if } |S| \neq 0 \text{ and } H_0(S) = 0 , \\ H_0(S) & \text{otherwise} . \end{cases}$$

For a dependency relation (P, c) with nonempty P, the *modified conditional empirical entropy* of c given P is then defined as

$$H_P^*(c) = \frac{1}{m} \sum_{\rho \in \mathcal{A}(T[P])} |\rho_c| H_0^*(\rho_c) ,$$

where ρ_c is the string formed by catenating the symbols in c corresponding to positions of ρ in $T[P]$ [14]. A possible choice of $\omega(P, c)$ is given by $H_P^*(c)$. Vo and Vo also develop another notion of entropy, called *run length entropy*, to approximate more effectively the compressibility of low-entropy columns and define another cost function ω accordingly.

Key Results

Combinatorial Dependency

Problem 1 admits a polynomial-time algorithm, based on dynamic programming. Using the definition of combinatorial dependency, one can show:

Theorem 1 ([2]) *Let $E[i]$ be the cost of an optimal, contiguous partition of $T[1, i]$. $E[n]$ is thus the cost of a solution to Problem 1. Define $E[0] = 0$; then, for $1 \leq i \leq n$,*

$$E[i] = \min_{0 \leq j < i} E[j] + H_C(T_{j+1}, \ldots, T_i) . \quad (1)$$

The actual partition with cost $E[n]$ can be maintained by standard backtracking.

The only known algorithmic solution to Problem 2 is the trivial one based on enumerating all possible feasible solutions to choose an optimal one. Some efficient heuristics based on asymmetric TSP, however, have been devised and tested experimentally [3]. Define a weighted, complete, directed graph, $G(T)$, with a vertex T_i for each column $T[i] \in T$; the *weight* of edge $\{T_i, T_j\}$ is $w(T_i, T_j) = \min(H_C(T_i, T_j), H_C(T_i) + H_C(T_j))$. One then generates a set of tours of various weights by iteratively applying standard optimizations (e. g., 3-opt, 4-opt). Each tour induces an ordering of the columns, which are then optimally partitioned using the dynamic program (1).

Buchsbaum et al. [3] also provide a general framework for studying the computational complexity of several variations of table compression problems based on notions analogous to combinatorial dependence, and they give some initial MAX-SNP-hardness results. Particularly relevant is the set of abstract problems in which one is required to find an optimal arrangement of a set of strings to be compressed, which establishes a nontrivial connection between table compression and the classical shortest common superstring problem [1]. Giancarlo et al. [10] connect table compression to the Burrows and Wheeler transform [4] by deriving the latter as a solution to an analog of Problem 2.

Column Dependency

Theorem 2 ([14,15]) *For $k \geq 2$, Problem 3 is NP-hard.*

Theorem 3 ([14,15]) *An optimum 1-transform for a table T can be found in $O(n^2\delta(m))$ time.*

Theorem 4 ([14,15]) *A 2-transform can be computed in $O(n^2\delta(m))$ time.*

Theorem 5 ([14]) *For any dependency relation (P, c) and some constant ϵ, $|C(\text{dt}_P(c))| \leq 5mH_p^*(c) + \epsilon$.*

Applications

Storage and transmission of alphanumeric tables.

Open Problems

All the techniques discussed use the general paradigms of context-dependent data rearrangement for compression boosting. It remains open to apply these paradigms to other domains, e. g., XML data [9,12], where high-level structures can be exploited, and to domains where pertinent structures are not known a priori.

Experimental Results

Buchsbaum et al. [2] showed that optimal partitioning alone (no column rearrangement) yielded about 55% better compression compared to gzip on telephone usage data, with small training sets. Buchsbaum et al. [3] experimentally supported the hypothesis that good TSP heuristics can effectively reorder the columns, yielding additional improvements of 5 to 20% relative to partitioning alone. They extended the data sets used to include other tables from the telecom domain as well as biological data. Vo and Vo [14,15] showed further 10 to 35% improvement over these combinatorial dependency methods on the same data sets.

Data Sets

Some of the data sets used for experimentation are public [3].

URL to Code

The pzip package, based on combinatorial dependency, is available at http://www.research.att.com/~gsf/pzip/pzip.html. The Vcodex package, related to invertible transforms, is available at http://www.research.att.com/~gsf/download/ref/vcodex/vcodex.html. Although for the time being Vcodex does not include procedures to compress tabular data, it is a useful toolkit for their development.

Cross References

► Binary Decision Graph
► Burrows–Wheeler Transform
► Dictionary-Based Data Compression
► Succinct Data Structures for Parentheses Matching
► Tree Compression and Indexing

Recommended Reading

1. Blum, A., Li, M., Tromp, J., Yannakakis, M.: Linear approximation of shortest superstrings. J. ACM **41**, 630–47 (1994)
2. Buchsbaum, A.L., Caldwell, D.F., Church, K.W., Fowler, G.S., Muthukrishnan, S.: Engineering the compression of massive tables: An experimental approach. In: Proc. 11th ACM-SIAM Symp. on Discrete Algorithms, 2000, pp. 175–84
3. Buchsbaum, A.L., Fowler, G.S., Giancarlo, R.: Improving table compression with combinatorial optimization. J. ACM **50**, 825–851 (2003)
4. Burrows, M., Wheeler, D.: A block sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation (1994)
5. Cilibrasi, R., Vitanyi, P.M.B.: Clustering by compression. IEEE Trans. Inf. Theory **51**, 1523–1545 (2005)
6. Cormack, G.: Data compression in a data base system. Commun. ACM **28**, 1336–1350 (1985)
7. Cover, T.M., Thomas, J.A.: Elements of Information Theory. Wiley Interscience, New York, USA (1990)
8. Ferragina, P., Giancarlo, R., Manzini, G., Sciortino, M.: Boosting textual compression in optimal linear time. J. ACM **52**, 688–713 (2005)
9. Ferragina, P., Luccio, F., Manzini, G., Muthukrishnan, S.: Structuring Labeled Trees for Optimal Succinctness, and beyond. In: Proc. 45th Annual IEEE Symposium on Foundations of Computer Science, 2005, pp. 198–207
10. Giancarlo, R., Sciortino, M., Restivo, A.: From first principles to the Burrows and Wheeler transform and beyond, via combinatorial optimization. Theor. Comput. Sci. (2007)
11. Li, M., Chen, X., Li, X., Ma, B., Vitanyi, P.M.B.: The similarity metric. IEEE Trans. Inf. Theory **50**, 3250–3264 (2004)
12. Liefke, H., Suciu, D.: XMILL: An efficient compressor for XML data. In: Proceedings of the 2000 ACM SIGMOD Int. Conf. on Management of Data, pp. 153–164. ACM, New York, USA (2000)
13. Lifshits, Y., Mozes, S., Weimann, O., Ziv-Ukelson, M.: Speeding up HMM decoding and training by exploiting sequence repetitions. Algorithmica to appear doi:10.1007/s00453-007-9128-0
14. Manzini, G.: An analysis of the Burrows–Wheeler transform. J. ACM **48**, 407–430 (2001)
15. Vo, B.D., Vo, K.-P.: Compressing table data with column dependency. Theor. Comput. Sci. **387**, 273–283 (2007)
16. Vo, B.D., Vo, K.-P.: Using column dependency to compress tables. In: DCC: Data Compression Conference, pp. 92–101. IEEE Computer Society TCC, Washington DC, USA (2004)
17. Vo., K.-P.: Compression as data transformation. In: DCC: Data Compression Conference. IEEE Computer Society TCC, pp. 403. Washington DCD, USA (2006)
18. Ziv, J., Lempel, A.: A universal algorithm for sequential data compression. IEEE Trans. Inf. Theory **23**, 337–343 (1977)
19. Ziv, J., Lempel, A.: Compression of individual sequences via variable length coding. IEEE Trans. Inf. Theory **24**, 530–536 (1978)

Tail Bounds for Occupancy Problems

1995; Kamath, Motwani, Palem, Spirakis

PAUL SPIRAKIS
Computer Engineering and Informatics, Research and Academic Computer Technology Institute, Patras University, Patras, Greece

Keywords and Synonyms

Balls and bins

Problem Definition

Consider a *random allocation* of m balls to n bins where each ball is placed in a bin chosen uniformly and independently. The properties of the resulting distribution of balls among bins have been the subject of intensive study in the probability and statistics literature [3,4]. In computer science, this process arises naturally in randomized algorithms and probabilistic analysis. Of particular interest is the *occupancy problem* where the random variable under consideration is the number of empty bins.

In this entry a series of bounds are presented (reminiscent of the Chernoff bound for binomial distributions) on the tail of the distribution of the number of empty bins; the tail bounds are successively tighter, but each new bound has a more complex closed form. Such strong bounds do not seem to have appeared in the earlier literature.

Key Results

The following notation in presenting sharp bounds on the tails of distributions. The notation $F \sim G$ will denote that $F = (1 + o(1))G$; further, $F \asymp G$ will denote that $\ln F \sim \ln G$. The proof that $f \asymp g$, is used for the purposes of later claiming that $2^f \asymp 2^g$. These asymptotic equalities will be treated like actual equalities and it will be clear that the results claimed are unaffected by this “approximation”.

Consider now the probabilistic experiment of throwing m balls, independently and uniformly, into n bins.

Definition 1 Let Z be the number of empty bins when m balls are placed randomly into n bins, and define $r = m/n$. Define the function $H(m, n, z)$ as the probability that $Z = z$. The expectation of Z is given by

$$\mu = \mathbf{E}[Z] = n\left(1 - \frac{1}{n}\right)^m \sim n\,\mathrm{e}^{-r} .$$

The following three theorems provide the bounds on the tail of the distribution of the random variable Z. The proof of the first bound is based on a martingale argument.

Theorem 1 (Occupancy Bound 1) *For any $\theta > 0$,*

$$\boldsymbol{P}\left[|Z - \mu| \geq \theta\mu\right] \leq 2\exp\left(-\frac{\theta^2\mu^2(n - \frac{1}{2})}{n^2 - \mu^2}\right) .$$

Remark that for large r this bound is asymptotically equal to

$$2\exp\left(-\frac{\theta^2\,\mathrm{e}^{-2r}n}{1 - \mathrm{e}^{-2r}}\right) .$$

The reader may wish to compare this with the following heuristic estimate of the tail probability assuming that the distribution of Z is well approximated by the approximating normal distribution also far out in the tails [3,4].

$$\boldsymbol{P}\left[|Z - \mu| \geq \theta\mu\right] \leq 2\exp\left(-\frac{\theta^2\,\mathrm{e}^{-r}n}{2\left(1 - (1 + r)\,\mathrm{e}^{-r}\right)}\right) .$$

The next two bounds are in terms of point probabilities rather than tail probabilities (as was the case in the Binomial Bound), but the unimodality of the distribution implies that the two differ by at most a small (linear) factor. These more general bounds on the point probability are essential for the application to the satisfiability problem. The next result is obtained via a generalization of the Binomial Bound to the case of dependent Bernoulli trials.

Theorem 2 (Occupancy Bound 2) *For $\theta > -1$,*

$$H(m, n, (1 + \theta)\mu) \leq \exp\left(-\left((1 + \theta)\ln[1 + \theta] - \theta\right)\mu\right) .$$

In particular, for $-1 \leq \theta < 0$,

$$H(m, n, (1 + \theta)\mu) \leq \exp\left(-\frac{\theta^2\mu}{2}\right) .$$

The last result is proved using ideas from large deviations theory [7].

Theorem 3 (Occupancy Bound 3) *For $|z - \mu| = \Omega(n)$,*

$$H(m, n, z) \asymp \exp\left(\left[-n\left(\int_0^{1-\frac{z}{n}} \ln\left[\frac{k - x}{1 - x}\right] \mathrm{d}x - r\ln k\right)\right]\right)$$

where k is defined implicitly by the equation $z = n(1 - k(1 - \mathrm{e}^{-r/k}))$.

Applications

Random allocations of balls to bins is a basic model that arises naturally in many areas in computer science involving choice between a number of resources, such as communication links in a network of processors, actuator devices in a wireless sensor network, processing units in a multi-processor parallel machine etc. For such situations, randomization can be used to “spread” the load evenly among the resources, an approach particularly useful in a parallel or distributed environment where resource utilization decisions have to be made locally at a large number of sites without reference to the global impact of these decisions. In the process of analyzing the performance of such algorithms, of particular interest is the occupancy problem where the random variable under consideration is the number of empty bins (i. e., machines with no jobs, routes with no load, etc.). The properties of the resulting distribution of balls among bins and the corresponding tails bounds may help in order to analyze the performance of such algorithms.

Cross References

- ► Approximation Schemes for Bin Packing
- ► Bin Packing

Recommended Reading

1. Kamath, A., Motwani, R., Spirakis, P., Palem, K.: Tail bounds for occupancy and the satisfiability threshold conjecture. J. Random Struct. Algorithms **7**(1), 59–80 (1995)

2. Janson, S.: Large Deviation Inequalities for Sums of Indicator Variables. Technical Report No. 34, Department of Mathematics, Uppsala University (1994)
3. Johnson, N.L., Kotz, S.: Urn Models and Their Applications. Wiley, New York (1977)
4. Kolchin, V.F., Sevastyanov, B.A., Chistyakov, V.P.: Random Allocations. Wiley, New York (1978)
5. Motwani, R., Raghavan, P.: Randomized Algorithms. Cambridge University Press, New York (1995)
6. Shwartz, A., Weiss, A.: Large Deviations for Performance Analysis. Chapman-Hall, Boca Raton (1994)
7. Weiss, A.: Personal Communication (1993)

Technology Mapping

1987; Keutzer

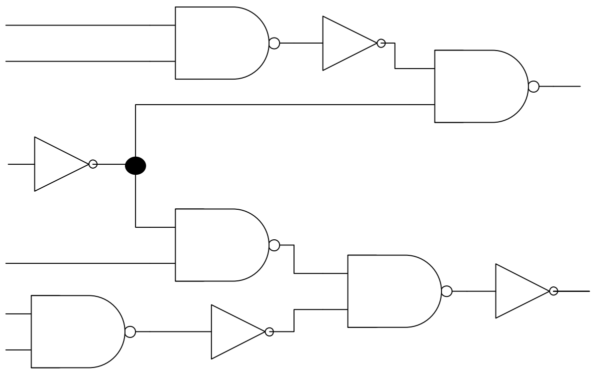
Kurt Keutzer, Kaushik Ravindran
Department of Electrical Engineering and Computer Science, University of California at Berkeley, Berkeley, CA, USA

Keywords and Synonyms

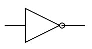
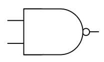
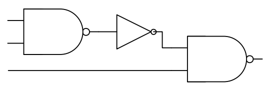
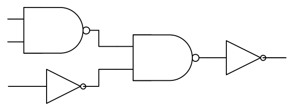
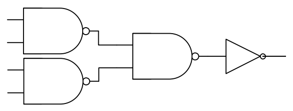
Library-based technology mapping; Technology dependent optimization

Problem Definition

Technology mapping is the problem of implementing a sequential circuit using the gates of a particular technology library. It is an integral component of any automated VLSI circuit design flow. In the prototypical chip design flow, combinational logic gates and sequential memory elements are composed to form sequential circuits. These circuits are subject to various logic optimizations to minimize area, delay, power and other performance metrics. The resulting optimized circuits still consist of primitive logic functions such as AND and OR gates. The next step is to efficiently realize these circuits in a specific VLSI technology using a library of gates available from the semiconductor vendor. Such a library would typically consist of gates of varying sizes and speeds for primitive logic functions, (AND and OR) and more complex functions (exclusive-OR, multiplexer). However, a naïve translation of generic logic elements to gates in the library will fall short of realistic performance goals. The challenge is to construct a mapping that maximally utilizes the gates in the library to implement the logic function of the circuit and achieve some performance goal—for example, minimum area with the critical path delay less than a target value. This is accomplished by *technology mapping*. For the sake of simplicity, in the following discussion it is presumed that the sequential memory elements are stripped from the digital circuit and mapped directly into memory



Technology Mapping, Figure 1
Subject graph (DAG) of a Boolean circuit expressed using NAND2 and INVERTER gates

Gate	Cost	Pattern Graph
INVERTER	2	
NAND2	3	
NAND3	4	
AND-OR-INVERT-21	4	
AND-OR-INVERT-22	5	

Technology Mapping, Figure 2
Library of pattern graphs (composed of NAND2 and INVERTER gates) and associated costs

elements of the particular technology. Then, only Boolean circuits composed of combinational logic gates remain to be mapped. Further, each remaining Boolean circuit is necessarily a directed acyclic graph (DAG).

The technology mapping problem can be restated in a more general graph-theoretic setting: *find a minimum cost covering of the subject graph (Boolean circuit) by choosing from the collection of pattern graphs (gates) available in a library*. The inputs to the problem are:

(a) *Subject graph*: This is a directed acyclic graph representation of a Boolean circuit expressed using a set of primitive functions (e. g., 2-input NAND gates and inverters). An example subject graph is shown in Fig. 1.

(b) *Library of pattern graphs*: This is a collection of gates available in the technology library. The pattern graphs are also DAGs expressed using the same primitive functions used to construct the subject graph. Additionally, each gate is annotated with a number of values for different cost functions, such as area, delay, and power. An example library and associated cost model is shown in Fig. 2.

A *valid cover* is a network of pattern graphs implementing the function of the subject graph such that: (a) every vertex (i. e. gate) of the subject graph is contained in some pattern graph, and (b) each input required by a pattern graph is actually an output of some other pattern graph (i. e. the inputs of a gate must exist as outputs of other gates). Technology mapping can then be viewed as an optimization problem to find a valid cover of minimum cost of the subject graph.

Key Results

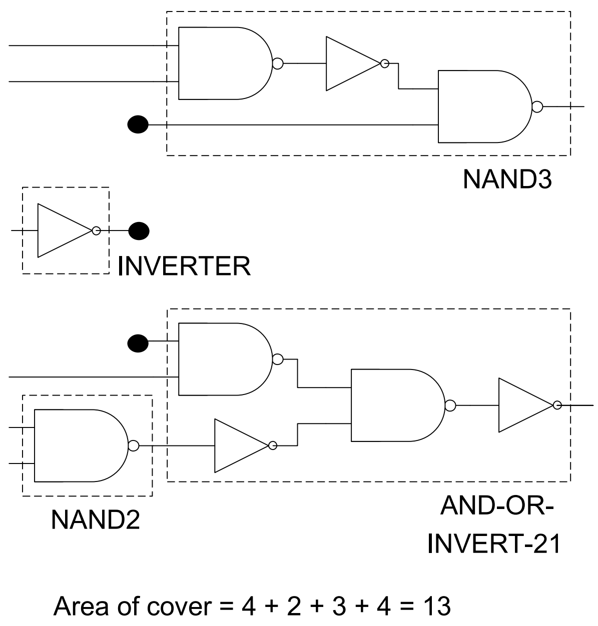
To be viable in a realistic design flow, an algorithm for minimum cost graph covering for technology mapping should ideally possess the following characteristics: (a) the algorithm should be easily adaptable to diverse libraries and cost models—if the library is expanded or replaced, the algorithm must be able to utilize the new gates effectively, (b) it should allow detailed cost models to accurately represent the performance of the gates in the library, and (c) it should be fast and robust on large subject graph instances and large libraries. One technique for solving the minimum cost graph covering problem is to formulate it as a binate-covering problem, which is a specialized integer linear program [5]. However, binate covering for a DAG is NP-Hard for any set of primitive functions and is typically unwieldy on large circuits. The DAGON algorithm suggested solving the technology mapping problem through DAG covering and advanced an alternate approach for DAG covering based on a *tree covering* approximation that produced near-optimal solutions for practical circuits and was very fast even for large circuits and large libraries [4].

DAGON was inspired by prevalent techniques for pattern matching employed in the domain of code generation for programming language compilers [1]. The fundamental concept was to partition the subject graph (DAG) into a forest of trees and solve the minimum cost covering problem independently for each tree. The approach was motivated by the existence of efficient dynamic programming algorithms for optimum tree covering [2]. The three salient components of the DAGON algorithm are: (a) subject graph partitioning, (b) pattern matching, and (c) covering.

(a) Subject graph partitioning: To apply the tree covering approximation the subject graph is first partitioned into a forest of trees. One approach is to break the graph at each vertex which has an out-degree greater than 1 (multiple fan-out point). The root of each tree is the primary output of the corresponding sub-circuit and the leaves are the primary inputs. Other heuristic partitions of the subject graph that consider duplication of vertices can also be applied to improve the quality of the final cover. Alternate subject graph partitions can also be derived starting from different decompositions of the original Boolean circuit in terms of the primitive functions.

(b) Pattern matching: The optimum covering of a tree is determined by generating the complete set of matches for each vertex in the tree (i. e. the set of pattern graphs which are candidates for covering a particular vertex) and then selecting the optimum match from among the candidates. An efficient approach for structural pattern matching is to reduce the tree matching problem to a *string matching* problem [2]. Fast string matching algorithms, such as the Aho–Corasick and the Knuth–Morris–Pratt algorithms, can then be used to find all strings (pattern graphs) which match a given vertex in the subject graph in time proportional to the length of the longest string in the set of pattern graphs. Alternatively, Boolean matching techniques can be used to find matches based on logic functions [12]. Boolean matching is slower than structural string matching, but it can compute matches independent of the actual local decompositions and under different input permutations.

(c) Covering: The final step is to generate a valid cover of the subject tree using the pattern graph matches computed at each vertex. Consider the problem of finding a valid cover of minimum area for the subject tree. Every pattern graph in the library has an associated area and the area of a valid cover is the sum of the area of the pattern graphs in the cover. The key property that makes minimum area tree covering efficient is this: *the minimum area cover of a tree rooted at some vertex v can be computed using only the minimum area covers of vertices below v*. If follows that for every pattern graph that matches at vertex v, the area of the minimum cover containing that match equals the sum of the area of the corresponding match at v and the sum of the areas of the optimal covers of the vertices which are inputs to that match. This property enables a dynamic programming algorithm to compute the minimum area cover of tree rooted at each vertex of the subject tree. The base case is the minimum area cover of a leaf (primary input) of subject tree. The area of a match at a leaf is set to 0. A recursive formulation of this dynamic programming concept is summarized in the Algorithm `minimum_area_tree_cover` shown below. As an example, the minimum area cover displayed in Fig. 3 is



Technology Mapping, Figure 3
Result of a minimum area tree covering of the subject graph in Fig. 1 using the library of pattern graphs in Fig. 2

a result of applying this algorithm to the tree partitions of the subject graph from Fig. 1 using the library from Fig. 2.

Given a vertex v in the subject tree, let M(v) denote the set of candidate matches from the library of pattern graphs for the sub-tree rooted at v.

```
Algorithm minimum_area_tree_cover (
       Vertex v ) {
   // the algorithm minimum_area_tree_cover
   //    finds an optimal cover of the tree
   //    rooted at Vertex v
   // the algorithm computes best_match(v)
   //    and areas_of_best_match(v), which
   //    denote the best pattern graph match
   //    at v and the associated areas of
   //    the optimal cover of the tree rooted
   //    at v respectively

   // check if v is a leaf of the tree
   if ( v is a leaf) {
      area_of_best_match(v) = 0;
      best_match(v) = leaf;
      return;
   }

   // compute optimal cover for each input
   //    of v
   foreach ( input of Vertex v ) {
       minimum_area_tree_cover( input );
   }
   // each tree rooted at each input of v is
   //    now annotated with its optimal cover

   // find the optimal cover of the tree
   //    rooted at Vertex v
   area_of_best_match(v) = INFINITY;
   best_match(v) = NULL;

   foreach ( Match m in the set of matches
       M(v) ) {
      // compute the area of match m at
      //    Vertex v
      // area_of_match(v,m) denotes the area
      //    of the cover when Match m is
      //    selected for v
      area_of_match(v,m) = area(m);
      foreach input pin v_i of matche m {
         area_of_match (v,m) =
            area_of_match(v,m) +
               area_of_best_match(v_i);
      }

      // update best pattern graph match
      //    and associated area of the optimal
      //    cover at Vertex v
      if ( area_of_match(v,m) <
          area_of_best_match(v) ) {
            area_of_best_match(v) =
               area_of_match(v,m);
            best_match(v) = m;
      }

   }
}
```

In this algorithm each vertex in the tree is visited exactly once. Hence, the complexity of the algorithm is proportional to the number of vertices in the subject tree times the maximum number of pattern matches at any vertex. The maximum number of matches is a function of the pattern graph library and is independent of the subject tree size. As a result, the complexity of computing the minimum cost valid cover of a tree is linear in the size of the subject tree, and the memory requirements are also linear in the size of the subject tree. The algorithm computes the optimum cover when the subject graph is a tree. In the general case of the subject graph being a DAG, empirical results have shown that the tree covering approximation yields industrial-quality results achieving aggressive area and timing requirements on large real circuit design problems [11,13].

Applications

Technology mapping is the key link between technology independent logic synthesis and technology dependent physical design of VLSI circuits. This motivates the need for efficient and robust algorithms to implement large Boolean circuits in a technology library. Early algorithms for technology mapping were founded on rule-based local transformations [3]. DAGON was the first in advancing an algorithmic foundation in terms of graph transformations that was practicable in the inner loop of iterative procedures in the VLSI design flow [4]. From a theoretical standpoint, the graph covering formulation provided a formal description of the problem and specified optimality criteria for evaluating solutions. The algorithm was naturally adaptable to diverse libraries and cost models, and was relatively easy to implement and extend. The concept of partitioning the subject graph into trees and covering the trees optimally was effective for varied optimization objectives such as area, delay, and power. The DAGON approach has been incorporated in academic (SIS from the University of California at Berkeley [6]) and industrial (Synopsys™ Design Compiler) tool offerings for logic synthesis and optimization.

The graph covering formulation has also served as a starting point for advancements in algorithms for technology mapping over the last decade. Decisions related to logic decomposition were integrated in the graph covering algorithm, which in turn enabled technology independent logic optimizations in the technology mapping phase [9]. Similarly, heuristics were proposed to impose placement constraints and make technology mapping more aware of the physical design and layout of the final circuit [10]. To combat the problem of high power dissipation in modern submicron technologies, the graph algorithms were enhanced to minimize power under area and delay constraints [8]. Specializations of these graph algorithms for technology mapping have found successful application in design flows for Field Programmable Gate Array (FPGA) technologies [7]. We recommend the following works for a comprehensive treatment of algorithms for technology mapping and a survey of new developments and challenges in the design of modern VLSI circuits: [11,12,13].

Open Problems

The enduring problem with DAGON-related technology mappers is handling non-tree pattern graphs that arise from modeling circuit elements such as multiplexors, Exclusive-Ors, or memory-elements (e. g. flip-flops) with associated logic (e. g. scan logic). On the other hand, approaches that do not use the tree-covering formulation face challenges in easily representing diverse technology libraries and in matching the subject graph in a computationally efficient manner.

Cross References

► Sequential Exact String Matching

Recommended Reading

1. Aho, A., Sethi, R., Ullman, J.: Compilers: Principles, Techniques and Tools. pp. 557–584. Addison Wesley, Boston (1986)
2. Aho, A., Johnson, S.: Optimal Code Generation for Expression Trees. J. ACM **23**(July), 488–501 (1976)
3. Darringer, J.A., Brand, D., Gerbi, J.V., Joyner, W.H., Trevillyan, L.H.: LSS: Logic Synthesis through Local Transformations. IBM J. Res. Dev. **25**, 272–280 (1981)
4. Keutzer, K.: DAGON: Technology Binding and Local Optimizations by DAG Matching. In: Proc. of the 24th Design Automation Conference **28**(1), pp. 341–347. Miami Beach, June 1987
5. Rudell, R.: Logic Synthesis for VLSI Design. Ph. D. thesis, University of California at Berkeley, ERL Memo 89/49, April 1989
6. Sentovich, E.M., Singh, K.J., Moon, C., Savoj, H., Brayton, R.K., Sangiovanni-Vincentelli, A.: Sequential Circuit Design using Synthesis and Optimization. In: Proc. of the IEEE International Conference on Computer Design: VLSI in Computers & Processors (ICCD), pp. 328–333. Cambridge, October 1992
7. Cong, J., Ding, Y.: An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table based FPGA Designs. In: Proc. of the 1992 IEEE/ACM International Conference on Computer-Aided Design (ICCAD-92) **8**(12), pp. 48–53, November 1992
8. Tiwari, V., Ashar, P., Malik, S.: Technology Mapping for Low Power in Logic Synthesis. Integr. VLSI J. **20**(3), 243–268 (1996)
9. Lehman, E., Watanabe, Y., Grodstein, J., Harkness, H.: Logic Decomposition during Technology Mapping. IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. **16**(8), 813–834, (1997)
10. Kutzschebauch, T., Stok, L.: Congestion Aware Layout Driven Logic Synthesis. In: Proc. of the IEEE/ACM International Conference on Computer-Aided Design, 2001, pp. 216–223
11. Devadas, S., Ghosh, A., Keutzer, K.: Logic Synthesis. McGraw Hill, New York (1994). pp. 185–200
12. De Micheli, G.: Synthesis and Optimization of Digital Circuits, 1st edn., pp. 504–533. McGraw-Hill, New York (1994)
13. Stok, L., Tiwari, V.: Technology Mapping. In: Hassoun, S., Sasou, T. (eds.) Logic Synthesis and Verification, pp. 115–139. Kluwer International Series In Engineering And Coputer Science Series. Kluwer Academic Publisher, Norwell (2002)

Teleportation of Quantum States

1993; Bennett, Brassard, Crepeau, Jozsa, Peres, Wootters

RAHUL JAIN
Computer Science and Institute for Quantum Computing, University of Waterloo, Waterloo, ON, Canada

Keywords and Synonyms

Quantum teleportation; Teleportation

Problem Definition

An *n-qubit quantum state* is a positive semi-definite operator of unit trace in the complex Hilbert space $\mathbb{C}^{2^n}$. A *pure* quantum state is a quantum state with a unique non-zero eigenvalue. A pure state is also often represented by the unique unit eigenvector corresponding to the unique non-zero eigenvalue. In this article the standard (ket, bra) notation is followed as is often used in quantum mechanics, in which $|v\rangle$ (called as 'ket v') represents a column vector and $\langle v|$ (called as 'bra v') represents its conjugate transpose. A *classical n-bit* state is simply a probability distribution on the set $\{0,1\}^n$.

Let $\{|0\rangle, |1\rangle\}$ be the standard basis for $\mathbb{C}^2$. For simplicity of notation $|0\rangle \otimes |0\rangle$ are represented as $|0\rangle|0\rangle$ or simply $|00\rangle$. Similarly $|0\rangle\langle 0|$ represents $|0\rangle \otimes \langle 0|$. An EPR pair is a special two-qubit quantum state defined as $|\psi\rangle \triangleq \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$. It is one of the four *Bell states* which form a basis for $\mathbb{C}^4$.

Suppose there are two spatially separated parties Alice and Bob and Alice wants to send an arbitrary n-qubit quantum state ρ to Bob. Since classical communication is much more reliable, and possibly cheaper, than quantum communication, it is desirable that this task be achieved by communicating just classical bits. Such a procedure is referred to as *teleportation*.

Unfortunately, it is easy to argue that this is in fact not possible if arbitrary quantum states need to be communicated faithfully. However Bennett, Brassard, Crepeau, Jozsa, Peres, Wootters [2] presented the following nice solution to it.

Key Results

Alice and Bob are said to share an EPR pair if each hold one qubit of the pair. In this article a standard notation is followed in which classical bits are called 'cbits' and shared EPR pairs are called 'ebits'. Bennett et al. showed the following:

Theorem 1 *Teleportation of an arbitrary n-qubit state can be achieved with 2n cbits and n ebits.*

These shared EPR pairs are referred to as *prior entanglement* to the protocol since they are shared at the beginning of the protocol (before Alice gets her input state) and are independent of Alice's input state. This solution is a good compromise since it is conceivable that Alice and Bob share several EPR pairs at the beginning, when they are possibly together, in which case they do not require a quantum channel. Later they can use these EPR pairs to transfer several quantum states when they are spatially separated.

Now see how Bennett el al. [2] achieve teleportation. First note that in order to show Theorem 1 it is enough to show that a single qubit, which is possibly a part of a larger state ρ can be teleported, while preserving its entanglement with the rest of the qubits of ρ, using 2 cbits and 1 ebit. Also note that the larger state ρ can now be assumed to be a pure state without loss of generality.

Theorem *Let $|\phi\rangle_{AB} = a_0|\phi_0\rangle_{AB}|0\rangle_A + a_1|\phi_1\rangle_{AB}|1\rangle_A$, where a_0, a_1 are complex numbers with $|a_0|^2 + |a_1|^2 = 1$. Subscripts A, B (representing Alice and Bob respectively) on qubits signify their owner.*

It is possible for Alice to send two classical bits to Bob such that at the end of the protocol the final state is $a_0|\phi_0\rangle_{AB}|0\rangle_B + a_1|\phi_1\rangle_{AB}|1\rangle_B$.

Proof For simplicity of notation, let us assume below that $|\phi_0\rangle_{AB}$ and $|\phi_1\rangle_{AB}$ do not exist. The proof is easily modified when they do exist by tagging them along. Let an EPR pair $|\psi\rangle_{AB} = \frac{1}{\sqrt{2}}(|0\rangle_A|0\rangle_B + |1\rangle_A|1\rangle_B)$ be shared between Alice and Bob. Let us refer to the qubit under concern that needs to be teleported as the input qubit.

The combined starting state of all the qubits is

$$\begin{aligned}|\theta_0\rangle_{AB} &= |\phi\rangle_{AB}|\psi\rangle_{AB}\\ &= (a_0|0\rangle_A + a_1|1\rangle_A)\left(\frac{1}{\sqrt{2}}(|0\rangle_A|0\rangle_B + |1\rangle_A|1\rangle_B)\right)\end{aligned}$$

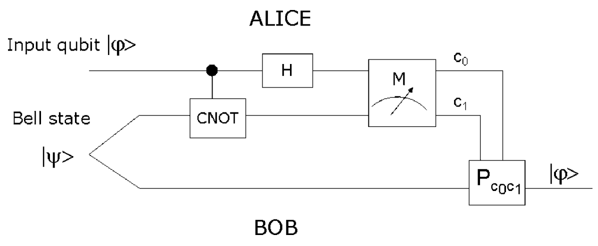
Let CNOT (*controlled-not*) gate be a two-qubit unitary operation described by the operator $|00\rangle\langle 00| + |01\rangle\langle 01| + |11\rangle\langle 10| + |10\rangle\langle 11|$. Alice now performs a CNOT gate on the input qubit and her part of the shared EPR pair. The resulting state is then,

$$\begin{aligned}|\theta_1\rangle_{AB} = &\frac{a_0}{\sqrt{2}}|0\rangle_A(|0\rangle_A|0\rangle_B + |1\rangle_A|1\rangle_B)\\ &+ \frac{a_1}{\sqrt{2}}|1\rangle_A(|1\rangle_A|0\rangle_B + |0\rangle_A|1\rangle_B)\ .\end{aligned}$$

Let the Hadamard transform be a single qubit unitary operation with operator $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\langle 0| + \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)\langle 1|$. Alice next performs a Hadamard transform on her input qubit. The resulting state then is,

$$\begin{aligned}|\theta_2\rangle_{AB} = &\frac{a_0}{2}(|0\rangle_A + |1\rangle_A)(|0\rangle_A|0\rangle_B + |1\rangle_A|1\rangle_B)\\ &+ \frac{a_1}{2}(|0\rangle_A - |1\rangle_A)(|1\rangle_A|0\rangle_B + |0\rangle_A|1\rangle_B)\\ = &\tfrac{1}{2}(|00\rangle_A(a_0|0\rangle_B + a_1|1\rangle_B) + |01\rangle_A(a_0|1\rangle_B\\ &+a_1|0\rangle_B)) + \tfrac{1}{2}(|10\rangle_A(a_0|0\rangle_B - a_1|1\rangle_B)\\ &+|11\rangle_A(a_0|1\rangle_B - a_1|0\rangle_B))\end{aligned}$$

Alice next measures the two qubits in her possession in the standard basis for $\mathbb{C}^4$ and sends the result of the measurement to Bob.



Teleportation of Quantum States, Figure 1
Teleportation protocol. H represent Hadamard transform and M represents measurement in the standard basis for $\mathbb{C}^4$

Let the four Pauli gates be the single qubit unitary operations: Identity: $P_{00} = |0\rangle\langle 0| + |1\rangle\langle 1|$, bit flip: $P_{01} = |1\rangle\langle 0| + |0\rangle\langle 1|$, phase flip: $P_{10} = |0\rangle\langle 0| - |1\rangle\langle 1|$ and bit flip together with phase flip: $P_{11} = |1\rangle\langle 0| - |0\rangle\langle 1|$. On receiving the two bits c_0c_1 from Alice, Bob performs the Pauli gate $P_{c_0c_1}$ on his qubit. It is now easily verified that the resulting state of the qubit with Bob would be $a_0|0\rangle_B + a_1|1\rangle_B$. The input qubit is successfully teleported from Alice to Bob! Please refer to Fig. 1 for the overall protocol. □

Super-Dense Coding

Super-dense coding [11] protocol is a dual to the teleportation protocol. In this Alice transmits 2 cbits of information to Bob using 1 qubit of communication and 1 shared ebit. It is discussed more elaborately in another article in the encyclopedia.

Lower Bounds on Resources

The above implementation of teleportation requires 2 cbits and 1 ebit for teleporting 1 qubit. It was argued in [2] that these resource requirements are also independently optimal. That is 2 cbits need to be communicated to teleport a qubit independent of how many ebits are used. Also 1 ebit is required to teleport one qubit independent of how much (possibly two-way) communication is used.

Remote State Preparation

Closely related to the problem of teleportation is the problem of *Remote state preparation* (RSP) introduced by Lo [10]. In teleportation Alice is just given the state to be teleported in some input register and has no other information about it. In contrast, in RSP, Alice knows a *complete description* of the input state that needs to be teleported. Also in RSP, Alice is not required to maintain any correlation of the input state with the other parts of a possibly larger state as is achieved in teleportation. The extra knowledge that Alice possesses about the input state can be used to devise protocols for probabilistically exact RSP with one cbit and one ebit per qubit asymptotically [3]. In a probabilistically exact RSP, Alice and Bob can abort the protocol with a small probability, however when they do not abort, the state produced with Bob at the end of the protocol, is exactly the state that Alice intends to send.

Teleportation as a Private Quantum Channel

The teleportation protocol that has been discussed in this article also satisfies an interesting privacy property. That is if there was a third party, say Eve, having access to the communication channel between Alice and Bob, then Eve learns nothing about the input state of Alice that she is teleporting to Bob. This is because the distribution of the classical messages of Alice is always uniform, independent of her input state. Such a channel is referred to as a *Private quantum channel* [1,6,8].

Applications

Apart from the main application of transporting quantum states over large distances using only classical channel, the teleportation protocol finds other important uses as well. A generalization of this protocol to implement unitary operations [7], is used in *Fault tolerant computation* in order to construct an infinite class of fault tolerant gates in a uniform fashion. In another application, a form of teleportation called as the error correcting teleportation, introduced by Knill [9], is used in devising quantum circuits that are resistant to very high levels of noise.

Experimental Results

Teleportation protocol has been experimentally realized in various different forms. To name a few, by Boschi et al. [4] using optical techniques, by Bouwmeester et al. [5] using photon polarization, by Nielsen et al. [12] using *Nuclear magnetic resonance* (NMR) and by Ursin et al. [13] using photons for long distance.

Cross References

► Quantum Dense Coding

Recommended Reading

1. Ambainis, A., Mosca, M., Tapp, A., de Wolf, R.: Private quantum channels. In: Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science, 2000, pp. 547–553
2. Bennett, C., Brassard G., Crepeau, C., Jozsa, R., Peres, A., Wootters, W.: Teleporting an unknown quantum state via dual classical and einstein-podolsky-rosen channels. Phys. Rev. Lett. **70**, 1895–1899 (1993)

3. Bennett, C.H., Hayden, P., Leung, W., Shor, P.W., Winter, A.: Remote preparation of quantum states. IEEE Trans. Inform. Theory **51**, 56–74 (2005)
4. Boschi, D., Branca, S., Martini, F.D., Hardy, L., Popescu, S.: Experimental realization of teleporting an unknown pure quantum state via dual classical and einstein-podolski-rosen channels. Phys. Rev. Lett. **80**, 1121–1125 (1998)
5. Bouwmeester, D., Pan, J.W. , Mattle, K., Eible, M., Weinfurter, H., Zeilinger, A.: Experimental quantum teleportation. Nature **390**(6660), 575–579 (1997)
6. Boykin, P.O., Roychowdhury, V.: Optimal encryption of quantum bits. Phys. Rev. A **67**, 042317 (2003)
7. Chaung, I.L., Gottesman, D.: Quantum teleportation is a universal computational primitive. Nature **402**, 390–393 (1999)
8. Jain, R.: Resource requirements of private quantum channels and consequence for oblivious remote state preparation. Technical report (2005). arXive:quant-ph/0507075
9. Knill, E.: Quantum computing with realistically noisy devices. Nature **434**, 39–44 (2005)
10. Lo, H.-K.: Classical communication cost in distributed quantum information processing – a generalization of quantum communication complexity. Phys. Rev. A **62**, 012313 (2000)
11. Nielsen, M., Chuang, I.: Quantum Computation and Quantum Information. Cambridge University Press (2000)
12. Nielsen, M.A., Knill, E., Laflamme, R.: Complete quantum teleportation using nuclear magnetic resonance. Nature **396**(6706), 52–55 (1998)
13. Ursin, R., Jennewein, T., Aspelmeyer, M., Kaltenbaek, R., Lindenthal, M., Zeilinger, A.: Quantum teleportation link across the danube. Nature **430**(849), 849–849 (2004)

Text Indexing

1993; Manber, Myers

SRINIVAS ALURU
Department of Electrical and Computer Engineering, Iowa State University, Ames, IA, USA

Keywords and Synonyms

String indexing

Problem Definition

Text or string data naturally arises in many contexts including document processing, information retrieval, natural and computer language processing, and describing molecular sequences. In broad terms, the goal of text indexing is to design methodologies to store text data so as to significantly improve the speed and performance of answering queries. While text indexing has been studied for a long time, it shot into prominence during the last decade due to the ubiquity of web-based textual data and search engines to explore it, design of digital libraries for archiving human knowledge, and application of string techniques to further understanding of modern biology. Text indexing differs from the typical indexing of keys drawn from an underlying total order—text data can have varying lengths, and queries are often more complex and involve substrings, partial matches, or approximate matches.

Queries on text data are as varied as the diverse array of applications they support. Consequently, numerous methods for text indexing have been developed and this continues to be an active area of research. Text indexing methods can be classified into two categories: (i) methods that are generalizations or adaptations of indexing methods developed for an ordered set of one-dimensional keys, and (ii) methods that are specifically designed for indexing text data. The most classic query in text processing is to find all occurrences of a pattern P in a given text T (or equivalently, in a given collection of strings). Important and practically useful variants of this problem include finding all occurrences of P subject to at most k mismatches, or at most k insertions/deletions/mismatches. The focus in this entry is on these two basic problems and remarks on generalizations of one-dimensional data structures to handle text data.

Key Results

Consider the problem of finding a given pattern P in text T, both strings over alphabet Σ. The case of a collection of strings can be trivially handled by concatenating the strings using a unique end of string symbol, not in Σ, to create text T. It is worth mentioning the special case where T is structured—i. e., T consists of a sequence of words and the pattern P is a word. Consider a total order of characters in Σ. A string (or word) of length k can be viewed as a k-dimensional key and the order on Σ can be naturally extended to lexicographic order between multidimensional keys of variable length. Any one-dimensional search data structure that supports $O(\log n)$ search time can be used to index a collection of strings using lexicographic order such that a string of length k can be searched in $O(k \log n)$ time. This can be considerably improved as below [8]:

Theorem 1 *Consider a data structure on one-dimensional keys that relies on constant-time comparisons among keys (e. g., binary search trees, red-black trees etc.) and the insertion of a key identifies either its predecessor or successor. Let $O(\mathcal{F}(n))$ be the search time of the data structure storing n keys (e. g., $O(\log n)$ for red-black trees). The data structure can be converted to index n strings using $O(n)$ additional space such that the query for a string s can be performed in $O(\mathcal{F}(n))$ time if s is one of the strings indexed, and in $O(\mathcal{F}(n) + |s|)$ otherwise.*

A more practical technique that provides $O(\mathcal{F}(n) + |s|)$ search time for a string s under more restrictions on the underlying one-dimensional data structure is given in [9]. The technique is nevertheless applicable to several classic one-dimensional data structures, in particular binary search trees and its balanced variants. For a collection of strings that share long common prefixes such as IP addresses and XML path strings, a faster search method is described in [5].

When answering a sequence of queries, significant savings can be obtained by promoting frequently searched strings so that they are among the first to be encountered in a search path through the indexing data structure. Ciriani et al. [4] use self-adjusting skip lists to derive an expected bound for a sequence of queries that matches the information-theoretic lower bound.

Theorem 2 *A collection of n strings of total length N can be indexed in optimal $O(N)$ space so that a sequence of m string queries, say $s_1,\cdots,s_m$, can be performed in $O(\sum_{j=1}^{m} |s_j| + \sum_{i=1}^{n} n_i \log(m/n_i))$ expected time, where n_i is the number of times the ith string is queried.*

Notice that the first additive term is a lower bound for reading the input, and the second additive term is a standard information-theoretic lower bound denoting the entropy of the query sequence. Ciriani et al. also extended the approach to the external memory model, and to the case of dynamic sets of strings. More recently, Ko and Aluru developed a self-adjusting tree layout for dynamic sets of strings in secondary storage that provides optimal number of disk accesses for a sequence of string or substring queries, thus providing a deterministic algorithm that matches the information-theoretic lower bound [4].

The next part of this entry deals with some of the widely used data structures specifically designed for string data, suffix trees, and suffix arrays. These are particularly suitable for querying unstructured text data, such as the genomic sequence of an organism. The following notation is used: Let $s[i]$ denote the ith character of string s, $s[i..j]$ denote the substring $s[i]s[i+1]\ldots s[j]$, and $S_i = s[i]s[i+1]\ldots s[|s|]$ denote the suffix of s starting at ith position. The suffix S_i can be uniquely described by the integer i. In case of multiple strings, the suffix of a string can be described by a tuple consisting of the string number and the starting position of the suffix within the string. Consider a collection of strings over Σ, having total length n, each extended by adding a unique termination symbol $\$ \notin \Sigma$. The suffix tree of the strings is a compacted trie of all suffixes of these extended strings. The suffix array of the strings is the lexicographic sorted order of all suffixes of these extended strings. For convenience, we list '\$', the last suffix of each string, just once. The suffix tree and suffix array of strings 'apple' and 'maple' are shown in Fig. 1. Both these data structures take $O(n)$ space and can be constructed in $O(n)$ time [11, 13], both directly and from each other.

Without loss of generality, consider the problem of searching for a pattern P as a substring of a single string T. Assume the suffix tree ST of T is available. If P occurs in T starting from position i, then P is a prefix of suffix $T_i = T[i]T[i+1]\ldots T[|T|]$ in T. It follows that P matches the path from root to leaf labeled i in ST. This property results in the following simple algorithm: Start from the root of ST and follow the path matching characters in P, until P is completely matched or a mismatch occurs. If P is not fully matched, it does not occur in T. Otherwise, each leaf in the subtree below the matching position gives an occurrence of P. The positions can be enumerated by traversing the subtree in $O(occ)$ time, where occ denotes the number of occurrences of P. If only one occurrence is desired, ST can be preprocessed in $O(|T|)$ time such that each internal node contains the suffix at one of the leaves in its subtree.

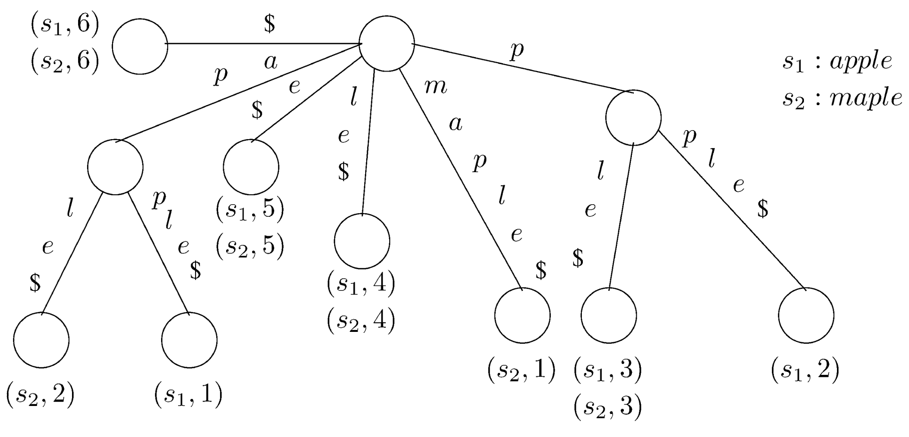
Theorem 3 *Given a suffix tree for text T and a pattern P, whether P occurs in T can be answered in $O(|P|)$ time. All occurrences of P in T can be found in $O(|P| + occ)$ time, where occ denotes the number of occurrences.*

Now consider solving the same problem using the suffix array SA of T. All suffixes prefixed by P appear in consecutive positions in SA. These can be found using binary search in SA. Naively performed, this would take $O(|P| * \log|T|)$ time. It can be improved to $O(|P| + \log|T|)$ time as follows [15]:

Let $SA[L..R]$ denote the range in the suffix array where the binary search is focused. To begin with, $L = 1$ and $R = |T|$. Let $\prec$ denote "lexicographically smaller", $\preceq$ denote "lexicographically smaller or equal", and $lcp(\alpha, \beta)$ denote the length of the longest common prefix between strings α and β. At the beginning of an iteration, $T_{SA[L]} \preceq P \preceq T_{SA[R]}$. Let $M = \lceil (L+R)/2 \rceil$. Let $l = lcp(P, T_{SA[L]})$ and $r = lcp(P, T_{SA[R]})$. Because SA is lexicographically ordered, $lcp(P, T_{SA[M]}) \geq \min(l, r)$. If $l = r$, then compare P and $T_{SA[M]}$ starting from the (l+1)th character. If $l \neq r$, consider the case when $l > r$.

Case I: $l < lcp(T_{SA[L]}, T_{SA[M]})$. In this case, $T_{SA[M]} \prec P$ and $lcp(P, T_{SA[M]}) = lcp(P, T_{SA[L]})$. Continue search in $SA[M..R]$. No character comparisons required.

Case II: $l > lcp(T_{SA[L]}, T_{SA[M]})$. In this case, $P \prec T_{SA[M]}$ and $lcp(P, T_{SA[M]}) = lcp(T_{SA[L]}, T_{SA[M]})$. Continue



SA										
$(s_1,6)$	$(s_2,2)$	$(s_1,1)$	$(s_1,5)$	$(s_2,5)$	$(s_1,4)$	$(s_2,4)$	$(s_2,1)$	$(s_1,3)$	$(s_2,3)$	$(s_1,2)$

Text Indexing, Figure 1
Suffix tree and suffix array of strings *apple* and *maple*

search in $SA[L..M]$. No character comparisons required.

Case III: $l = lcp(T_{SA[L]}, T_{SA[M]})$. In this case, $lcp(P, T_{SA[M]}) \geq l$. Compare P and $T_{SA[M]}$ beyond lth character to determine their relative order and *lcp*.

Similarly, the case when $r > l$ can be handled such that comparisons between P and $T_{SA[M]}$, if at all needed, start from $(r+1)$th character. To start the execution of the algorithm, $lcp(P, T_{SA[1]})$ and $lcp(P, T_{SA[|T|]})$ are computed directly using at most $2|P|$ character comparisons. It remains to be described how the $lcp(T_{SA[L]}, T_{SA[M]})$ and $lcp(T_{SA[R]}, T_{SA[M]})$ values required in each iteration are computed. Let $Lcp[1 \ldots |T| - 1]$ be an array such that $Lcp[i] = lcp(SA[i], SA[i+1])$. The *Lcp* array can be computed from *SA* in $O(|T|)$ time [12]. For any $1 \leq i < j \leq n$, $lcp(T_{SA[i]}, T_{SA[j]}) = \min_{k=i}^{j-1} Lcp[k]$. In order to find the *lcp* values required by the algorithm in constant time, note that the binary search can be viewed as traversing a path in the binary tree corresponding to all possible search intervals used by any execution of the binary search algorithm [15]. The root of the tree denotes the interval $[1..n]$. If $[i..j]$ $(j - i \geq 2)$ is the interval at an internal node of the tree, its left child is given by $[i..\lceil (i+j)/2 \rceil]$ and its right child is given by $[\lceil (i+j)/2 \rceil ..j]$. The *lcp* value for each interval in the tree is precomputed and recorded in $O(n)$ time and space.

Theorem 4 *Given the suffix array SA of text T and a pattern P, the existence of P in T can be checked in $O(|P| + \log|T|)$ time. All occurrences of P in T can be found in O(occ) additional time, where occ denotes their number.*

Proof The algorithm makes at most $2|P|$ comparisons in determining $lcp(P, T_{SA[1]})$ and $lcp(P, T_{SA[n]})$. A comparison made in an iteration to determine $lcp(P, T_{SA[M]})$ is categorized *successful* if it contributes the *lcp*, and categorized *failed* otherwise. There is at most one failed comparison per iteration. As for successful comparisons, note that the comparisons start with $(\max(l,r)+1)^{th}$ character of P, and each successful comparison increases the value of $\max(l,r)$ for the next iteration. Thus, each character of P is involved only once in a successful comparison. The total number of character comparisons is at most $3|P| + \log|T| = O(|P| + \log|T|)$. □

Abouelhoda et al. [1] reduce this time further to $O(|P|)$ by mimicking the suffix tree algorithm on a suffix array with some auxiliary information. The strategy is useful in other applications based on top-down traversal of suffix trees. At this stage, the distinction between suffix trees and suffix arrays is blurred as the auxiliary information stored makes the combined data structure equivalent to a suffix tree. Using clever implementation techniques, the space is reduced to approximately $6n$ bytes. A major advantage of the suffix tree and suffix array based methods is that the text T is often large and relatively static, while it is queried with several short patterns. With suffix trees and enhanced suffix arrays [1], once the text is preprocessed in $O(|T|)$ time, each pattern can be queried in $O(|P|)$ time for constant size alphabet. For large alphabets, the query can be answered in $O(|P| * \log|\Sigma|)$ time using $O(n|\Sigma|)$ space (by storing an ordered array of $|\Sigma|$ pointers to potential children of a node), or in $O(|P| * |\Sigma|)$ time using $O(n)$ space (by storing pointers to first child and next sibling).[1] For indexing in various text-dynamic situations, see [3,7] and references therein. The problem of compressing suffix trees and arrays is covered in more detail in other entries.

While exact pattern matching has many useful applications, the need for approximate pattern matching arises in several contexts ranging from information retrieval to finding evolutionary related biomolecular sequences. The classic approximate pattern matching problem is to find substrings in the text T that have an edit distance of k or less to the pattern P, i. e., the substring can be converted to P with at most k insert/delete/substitute operations. This problem is covered in more detail in other entries. Also see [16], the references therein, and Chapter 36 of [2].

Applications

Text indexing has many practical applications—finding words or phrases in documents under preparation, searching text for information retrieval from digital libraries, searching distributed text resources such as the web, processing XML path strings, searching for longest matching prefixes among IP addresses for internet routing, to name just a few. The reader interested in further exploring text indexing is referred to the book by Crochemore and Rytter [6], and to other entries in this Encyclopedia. The last decade of explosive growth in computational biology is aided by the application of string processing techniques to DNA and protein sequence data. String indexing and aggregate queries to uncover mutual relationships between strings are at the heart of important scientific challenges such as sequencing genomes and inferring evolutionary relationships. For an in depth study of such techniques, the reader is referred to Parts I and II of [10] and Parts II and VIII of [2].

Open Problems

Text indexing is a fertile research area, making it impossible to cover many of the research results or actively pursued open problems in a short amount of space. Providing better algorithms and data structures to answer a flow of string-search queries when caches or other query models are taken into account, is an interesting research issue [4].

[1]Recently, Cole et al. (2006) showed how to further reduce the search time to $O(|P| + \log|\Sigma|)$ while still keeping the optimal $O(|T|)$ space.

Cross References

▶ Compressed Suffix Array
▶ Compressed Text Indexing
▶ Indexed Approximate String Matching
▶ Suffix Array Construction
▶ Suffix Tree Construction in Hierarchical Memory
▶ Suffix Tree Construction in RAM
▶ Two-Dimensional Pattern Indexing

Recommended Reading

1. Abouelhoda, M., Kurtz, S., Ohlebusch, E.: Replacing suffix trees with enhanced suffix arrays. J. Discret. Algorithms **2**, 53–86 (2004)
2. Aluru, S. (ed.): Handbook of Computational Molecular Biology. Computer and Information Science Series. Chapman and Hall/CRC Press, Boca Raton (2005)
3. Amir, A., Kopelowitz, T., Lewenstein, M., Lewenstein, N.: Towards real-time suffix tree construction. In: Proc. String Processing and Information Retrieval Symposium (SPIRE), 2005, pp. 67–78
4. Ciriani, V., Ferragina, P., Luccio, F., Muthukrishnan, S.: A data structure for a sequence of string acesses in external memory. ACM Trans. Algorithms **3** (2007)
5. Crescenzi, P., Grossi, R., Italiano, G.: Search data structures for skewed strings. In: International Workshop on Experimental and Efficient Algorithms (WEA). Lecture Notes in Computer Science, vol. 2, pp. 81–96. Springer, Berlin (2003)
6. Crochemore, M., Rytter, W.: Jewels of Stringology. World Scientific Publishing Company, Singapore (2002)
7. Ferragina, P., Grossi, R.: Optimal On-Line Search and Sublinear Time Update in String Matching. SIAM J. Comput. **3**, 713–736 (1998)
8. Franceschini, G., Grossi, R.: A general technique for managing strings in comparison-driven data structures. In: Annual International Colloquium on Automata, Languages and Programming (ICALP), 2004
9. Grossi, R., Italiano, G.: Efficient techniques for maintaining multidimensional keys in linked data structures. In: Annual International Colloquium on Automata, Languages and Programming (ICALP), 1999, pp. 372–381
10. Gusfield, D.: Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology. Cambridge University Press, New York (1997)
11. Karkkainen, J., Sanders, P., Burkhardt, S.: Linear work suffix arrays construction. J. ACM **53**, 918–936 (2006)
12. Kasai, T., Lee, G., Arimura, H. et al.: Linear-time longest-common-prefix computation in suffix arrays and its applications. In: Proc. 12th Annual Symposium, Combinatorial Pattern Matching (CPM), 2001, pp. 181–192
13. Ko, P., Aluru, S.: Space efficient linear time construction of suffix arrays. J. Discret. Algorithms **3**, 143–156 (2005)
14. Ko, P., Aluru, S.: Optimal self-adjusting tree for dynamic string data in secondary storage. In: Proc. String Processing and Information Retrieval Symposium (SPIRE). Lect. Notes Comp. Sci. vol. 4726, pp. 184–194, Santiago, Chile (2007)
15. Manber, U., Myers, G.: Suffix arrays: a new method for on-line search. SIAM J. Comput. **22**, 935–948 (1993)

16. Navarro, G.: A guided tour to approximate string matching. ACM Comput. Surv. **33**, 31–88 (2001)

Thresholds of Random k-SAT

2002; Kaporis, Kirousis, Lalas

ALEXIS KAPORIS, LEFTERIS KIROUSIS
Department of Computer Engineering and Informatics, University of Patras, Patras, Greece

Keywords and Synonyms

Phase transitions; Probabilistic analysis of a Davis–Putnam heuristic

Problem Definition

Consider n Boolean variables $V = \{x_1, \ldots, x_n\}$ and the corresponding set of $2n$ literals $L = \{x_1, \overline{x}_1 \ldots, x_n, \overline{x}_n\}$. A k-clause is a disjunction of k literals of distinct underlying variables. A random formula $\phi_{n,m}$ in k Conjunctive Normal Form (k-CNF) is the conjunction of m clauses, each selected in a uniformly random and independent way amongst the $2^k \binom{n}{k}$ possible k-clauses on n variables in V. The density r_k of a k-CNF formula $\phi_{n,m}$ is the clauses-to-variables ratio m/n.

It was conjectured that for each $k \geq 2$ there exists a critical density r_k^* such that asymptotically almost all (a.a.a.) k-CNF formulas with density $r < r_k^*$ ($r > r_k^*$) are satisfiable (unsatisfiable, respectively). So far, the conjecture has been proved only for $k = 2$ [3,11]. For $k \geq 3$, the conjecture still remains open but is supported by experimental evidence [14] as well as by theoretical, but nonrigorous, work based on Statistical Physics [15]. The value of the putative threshold r_3^* is estimated to be around 4.27. Approximate values of the putative threshold for larger values of k have also been computed.

As far as rigorous results are concerned, Friedgut [10] proved that for each $k \geq 3$ there exists a sequence $r_k^*(n)$ such that for any $\epsilon > 0$, a.a.a. k-CNF formulas $\phi_{n,\lfloor (r_k^*(n)-\epsilon)n \rfloor}$ ($\phi_{n,\lceil (r_k^*(n)+\epsilon)n \rceil}$) are satisfiable (unsatisfiable, respectively). The convergence of the sequence $r_k^*(n)$, $n = 0, 1, \ldots$ for $k \geq 3$ remains open.

Let now

$$\begin{aligned} r_k^{*-} &= \underline{\lim}_{n\to\infty} r_k^*(n) \\ &= \sup\{r_k : \Pr[\phi_{n,\lfloor r_k n \rfloor} \text{ is satisfiable } \to 1]\} \end{aligned}$$

and

$$\begin{aligned} r_k^{*+} &= \overline{\lim}_{n\to\infty} r_k^*(n) \\ &= \inf\{r_k : \Pr[\phi_{n,\lceil r_k n \rceil} \text{is satisfiable} \to 0]\} \,. \end{aligned}$$

Obviously, $r_k^{*-} \leq r_k^{*+}$. Bounding from below (from above) r_k^{*-} (r_k^{*+}, respectively) with an as large as possible (as small as possible, respectively) bound has been the subject of intense research work in the past decade.

Upper bounds to r_k^{*+} are computed by counting arguments. To be specific, the standard technique is to compute the expected number of satisfying truth assignments of a random formula with density r_k and find an as small as possible value of r_k for which this expected value approaches zero. Then, by Markov's inequality, it follows that for such a value of r_k, a random formula $\phi_{n,\lceil r_k n \rceil}$ is unsatisfiable asymptotically almost always. This argument has been refined in two directions: First, considering not all satisfying truth assignments but a subclass of them with the property that a satisfiable formula always has a satisfying truth assignment in the subclass considered. The restriction to a judiciously chosen such subclass forces the expected value of the number of satisfying truth assignments to get closer to the probability of satisfiability, and thus leads to a better (smaller) upper bound r_k. However, it is important that the subclass should be such that the expected value of the number of satisfying truth assignments can be computable by the available probabilistic techniques.

Second, make use in the computation of the expected number of satisfying truth assignments of *typical* characteristics of the random formula, i. e. characteristics shared by a.a.a. formulas. Again this often leads to an expected number of satisfying truth assignments that is closer to the probability of satisfiability (non-typical formulas may contribute to the increase of the expected number). Increasingly better upper bounds to r_3^{*+} have been computed using counting arguments as above (see the surveys [6,13]). Dubois, Boufkhad and Mandler [7] proved $r_3^{*+} < 4.506$. The latter remains the best upper bound to date.

On the other hand, for fixed and small values of k (especially for $k = 3$) lower bounds to r_k^{*-} are usually computed by algorithmic methods. To be specific, one designs an algorithm that for an as large as possible r_k it returns a satisfying truth assignment for a.a.a. formulas $\phi_{n,\lfloor r_k n \rfloor}$. Such an r_k is obviously a lower bound to r_k^{*-}. The simpler the algorithm, the easier to perform the probabilistic analysis of returning a satisfying truth assignment for a given r_k, but the smaller the r_k's for which a satisfying truth assignment is returned asymptotically almost always. In this context, backtrack-free DPLL algorithms [4,5] of increasing sophistication were rigorously analyzed (see the surveys [2,9]). At each step of such an algorithm, a literal is set to TRUE and then a *reduced* formula is obtained by (i) deleting clauses where this literal appears and by (ii) deleting the negation of this literal from the clauses it appears. At steps at which 1-clauses exist (known as forced steps), the selection of the literal to be set to TRUE is made so as a 1-clause becomes satisfied. At the remaining steps (known as free steps), the selection of the literal to be set to TRUE is made according to a heuristic that characterizes the particular DPLL algorithm. A free step is followed by a round of consecutive forced steps. To facilitate the probabilistic analysis of DPLL algorithms, it is assumed that they never backtrack: if the algorithm ever hits a contradiction, i. e. a 0-clause is generated, it stops and reports failure, otherwise it returns a satisfying truth assignment. The previously best lower bound for the satisfiability threshold obtained by such an analysis was $3.26 < r_3^{*-}$ (Achlioptas and Sorkin [1]).

The previously analyzed such algorithms (with the exception of the `Pure Literal` algorithm [8]) at a free step take into account only the clause size where the selected literal appears. Due to this limited information exploited on selecting the literal to be set, the reduced formula in each step remains random conditional only on the current numbers of 3- and 2-clauses and the number of yet unassigned variables. This retention of "strong" randomness permits a successful probabilistic analysis of the algorithm in a not very complicated way. However, for $k = 3$ it succeeds to show satisfiability only for densities up to a number slightly larger than 3.26. In particular, in [1] it is shown that this is the optimal value that can be attained by such algorithms.

Key Results

In [12], a DPLL algorithm is described (and then probabilistically analyzed) such that each free step selects the literal to be set to TRUE taking into account its *degree* (i. e. its number of occurrences) in the current formula.

Algorithm `Greedy` [Section 4.A in 12]

The first variant of the algorithm is very simple: At each free step, a literal with the maximum number of occurrences is selected and set to TRUE. Notice that in this greedy variant, a literal is selected irrespectively of the number of occurrences of its negation. This algorithm successfully returns a satisfying truth assignment for a.a.a. formulas with density up to a number slightly larger than 3.42, establishing that $r_3^{*-} > 3.42$. Its simplicity, contrasted with the improvement over the previously obtained lower bounds, suggests the importance of analyzing heuristics that take into account degree information of the current formula.

Algorithm `CL` [Section 5.A in 12]

In the second variant, at each free step t, the degree of the negation $\overline{\tau}$ of the literal τ that is set to TRUE is also taken into account. Specifically, the literal to be set to TRUE is selected so as upon the completion of the round of forced steps that follow the free step t, the marginal expected increase of the flow from 2-clauses to 1-clauses per unit of expected decrease of the flow from 3-clauses to 2-clauses is minimized. The marginal expectation corresponding to each literal can be computed from the numbers of its positive and negative occurrences. More specifically, if m_i, $i = 2, 3$ equals the expected flow of i-clauses to $(i-1)$-clauses at each step of a round, and τ is the literal set to TRUE at the beginning of the round, then τ is chosen so as to minimize the ratio $|\frac{\triangle m_2}{\triangle m_3}|$ of the differences $\triangle m_2$ and $\triangle m_3$ between the beginning and the end of the round. This has as effect the bounding of the rate of generation of 1-clauses by the smallest possible number throughout the algorithm. For the probabilistic analysis to go through, we need to know for each i, j the number of literals with degree i whose negation has degree j. This heuristic succeeds in returning a satisfying truth assignment for a.a.a. formulas with density up to a number slightly larger than 3.52, establishing that $r_3^{*-} > 3.52$.

Applications

Some applications of SAT solvers include Sequential Circuit Verification, Artificial Intelligence, Automated deduction and Planning, VLSI, CAD, Model-checking and other type of formal verification. Recently, automatic SAT-based model checking techniques were used to effectively find attacks on security protocols.

Open Problems

The main open problem in the area is to formally show the existence of the threshold r_k^* for all (or at least some) $k \geq 3$. To rigorously compute upper and lower bounds better than the ones mentioned here still attracts some interest. Related results and problems arise in the framework of variants of the satisfiability problem and also the problem of colorability.

Cross References

- ► Backtracking Based k-SAT Algorithms
- ► Local Search Algorithms for kSAT
- ► Maximum Two-Satisfiability
- ► Tail Bounds for Occupancy Problems

Recommended Reading

1. Achioptas, D., Sorkin, G.B.: Optimal myopic algorithms for random 3-sat. In: 41st Annual Symposium on Foundations of Computer Science, pp. 590–600. IEEE Computer Society, Washington (2000)
2. Achlioptas, D.: Lower bounds for random 3-sat via differential equations. Theor. Comput. Sci. **265**(1–2), 159–185 (2001)
3. Chvátal, V., Reed, B.: Mick gets some (the odds are on his side). In: 33rd Annual Symposium on Foundations of Computer Science, pp. 620–627. IEEE Computer Society, Pittsburgh (1992)
4. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem-proving. Commun. ACM **5**, 394–397 (1962)
5. Davis, M., Putnam, H.: A computing procedure for quantification theory. J. Assoc. Comput. Mach. **7**(4), 201–215 (1960)
6. Dubois, O.: Upper bounds on the satisfiability threshold. Theor. Comput. Sci. **265**, 187–197 (2001)
7. Dubois, O., Boufkhad, Y., Mandler, J.: Typical random 3-sat formulae and the satisfiability threshold. In: 11th ACM-SIAM symposium on Discrete algorithms, pp. 126–127. Society for Industrial and Applied Mathematics, San Francisco (2000)
8. Franco, J.: Probabilistic analysis of the pure literal heuristic for the satisfiability problem. Annal. Oper. Res. **1**, 273–289 (1984)
9. Franco, J.: Results related to threshold phenomena research in satisfiability: Lower bounds. Theor. Comput. Sci. **265**, 147–157 (2001)
10. Friedgut, E.: Sharp thresholds of graph properties, and the *k*-sat problem. J. AMS **12**, 1017–1054 (1997)
11. Goerdt, A.: A threshold for unsatisfiability. J. Comput. Syst. Sci. **33**, 469–486 (1996)
12. Kaporis, A.C., Kirousis, L.M., Lalas, E.G.: The probabilistic analysis of a greedy satisfiability algorithm. Random Struct. Algorithms **28**(4), 444–480 (2006)
13. Kirousis, L., Stamatiou, Y., Zito, M.: The unsatisfiability threshold conjecture: the techniques behind upper bound improvements. In: A. Percus, G. Istrate, C. Moore (eds.) Computational Complexity and Statistical Physics, Santa Fe Institute Studies in the Sciences of Complexity, pp. 159–178. Oxford University Press, New York (2006)
14. Mitchell, D., Selman, B., Levesque, H.: Hard and easy distribution of sat problems. In: 10th National Conference on Artificial Intelligence, pp. 459–465. AAAI Press, Menlo Park (1992)
15. Monasson, R., Zecchina, R.: Statistical mechanics of the random *k*-sat problem. Phys. Rev. E **56**, 1357–1361 (1997)

Topology Approach in Distributed Computing

1999; Herlihy Shavit

MAURICE HERLIHY
Department of Computer Science, Brown University, Providence, RI, USA

Keywords and Synonyms

Wait-free renaming

Problem Definition

The application of techniques from Combinatorial and Algebraic Topology has been successful at solving a number of problems in distributed computing. In 1993, three independent teams [3,15,17], using different ways of generalizing the classical graph-theoretical model of distributed computing, were able to solve *set agreement* a long-standing open problem that had eluded the standard approaches. Later on, in 2004, journal articles by Herlihy and Shavit [15] and by Saks and Zaharoglou [17] were to win the prestigious Gödel prize. This paper describes the approach taken by the Herlihy/Shavit paper, which was the first draw the connection between Algebraic and Combinatorial Topology and Distributed Computing.

Pioneering work in this area, such as by Biran, Moran, and Zaks [2] used graph-theoretic notions to model uncertainty, and were able to express certain lower bounds in terms of graph connectivity. This approach, however, had limitations. In particular, it proved difficult to capture the effects of multiple failures or to analyze decision problems other then consensus.

Combinatorial topology generalizes the notion of a graph to the notion of a *simplicial complex*, a structure that has been well-studied in mainstream mathematics for over a century. One property of central interest to topologists is whether a simplicial complex has no "holes" below a certain dimension *k*, a property known as *k-connectivity*. Lower bounds previously expressed in terms of connectivity of graphs can be generalized by recasting them in terms of *k*-connectivity of simplicial complexes. By exploiting this insight, it was possible to solve some open problems (*k*-set agreement, renaming), to pose and solve some new problems ([13]), and to unify a number of disparate results and models [14].

Key Results

A *vertex* $\vec{v}$ is a point in a high-dimensional Euclidean space. Vertexes $\vec{v}_0, \ldots, \vec{v}_n$ are *affinely independent* if $\vec{v}_1 - \vec{v}_0, \ldots, \vec{v}_n - \vec{v}_0$ are linearly independent. An *n-dimensional simplex* (or *n-simplex*) $S^n = (\vec{s}_0, \ldots, \vec{s}_n)$ is the convex hull of a set of $n+1$ affinely-independent vertexes. For example, a 0-simplex is a vertex, a 1-simplex a line segment, a 2-simplex a solid triangle, and a 3-simplex a solid tetrahedron. Where convenient, superscripts indicate dimensions of simplexes. The $\vec{s}_0, \ldots, \vec{s}_n$ are said to *span* S^n. By convention, a simplex of dimension $d < 0$ is an empty simplex.

A *simplicial complex* (or complex) is a set of simplexes closed under containment and intersection. The *dimension* of a complex is the highest dimension of any of its simplexes. $\mathcal{L}$ is a *subcomplex* of $\mathcal{K}$ if every simplex of $\mathcal{L}$ is a simplex of $\mathcal{K}$. A map $\mu\colon \mathcal{K} \to \mathcal{L}$ carrying vertexes to vertexes is *simplicial* if it also induces a map of simplexes to simplexes.

Definition 1 A complex $\mathcal{K}$ is *k-connected* if every continuous map of the k-sphere to $\mathcal{K}$ can be extended to a continuous map of the $(k+1)$-disk. By convention, a complex is (-1)-*connected* if and only if it is nonempty, and every complex is k-*connected* for $k < -1$.

A complex is 0-connected if it is connected in the graph-theoretic sense, and a complex is k-connected if it has no holes in dimensions k or less. The definition of k-connectivity may appear difficult to use, but fortunately reasoning about connectivity can be done in a combinatorial way, using the following elementary consequence of the Mayer–Vietoris sequence.

Theorem 2 *If $\mathcal{K}$ and $\mathcal{L}$ are complexes such that $\mathcal{K}$ and $\mathcal{L}$ are k-connected, and $\mathcal{K} \cap \mathcal{L}$ is $(k-1)$-connected, then $\mathcal{K} \cup \mathcal{L}$ is k-connected.*

This theorem, plus the observation that any non-empty simplex is k-connected for all k, allows reasoning about a complex's connectivity inductively in terms of the connectivity of its components.

A set of $n+1$ sequential *processes* communicate either by sending messages to one another or by applying operations to shared objects. At any point, a process may *crash*: it stops and takes no more steps. There is a bound f on the number of processes that can fail. Models differ in their assumptions about timing. At one end of the spectrum is the *synchronous model* in which computation proceeds in a sequence of rounds. In each round, a process sends messages to the other processes, receives the messages sent to it by the other processes in that round, and changes state. (Or it applies operations to shared objects.) All processes take steps at exactly the same rate, and all messages are delivered with exactly the same message delivery time. At the other end is the *asynchronous model* in which there is no bound on the amount of time that can elapse between process steps, and there is no bound on the time it can take for a message to be delivered. Between these extremes is the *semi-synchronous model* in which process step times and message delivery times can vary, but are bounded between constant upper and lower bounds. Proving a lower bound in any of these models requires a deep understanding of the global states that can arise in the course of a protocol's execution, and of how these global states are related.

Each process starts with an *input value* taken from a set V, and then executes a deterministic *protocol* in which it repeatedly receives one or more messages, changes its local state, and sends one or more messages. After a finite number of steps, each process chooses a *decision value* and halts.

In the k-set agreement task [5], processes are required to (1) choose a decision value after a finite number of steps, (2) choose as their decision values some process's input value, and (3) collectively choose no more than k distinct decision values. When $k = 1$, this problem is usually called *consensus* [16].

Here is the connection between topological models and computation. An initial local state of process P is modeled as a vertex $\vec{v} = \langle P, v\rangle$ labeled with P's process id and initial value v. An initial global state is modeled as an n-simplex $S^n = (\langle P_0, v_0\rangle, \ldots, \langle P_n, v_n\rangle)$, where the P_i are distinct. The term $ids(S^n)$ denotes the set of process ids associated with S^n, and $vals(S^n)$ the set of values. The set of all possible initial global states forms a complex, called the *input complex*.

Any protocol has an associated *protocol complex* $\mathcal{P}$, defined as follows. Each vertex is labeled with a process id and a possible local state for that process. A set of vertexes $\langle P_0, v_0\rangle, \ldots, \langle P_d, v_d\rangle$ spans a simplex of $\mathcal{P}$ if and only if there is some protocol execution in which $P_0, \ldots, P_d$ finish the protocol with respective local states $v_0, \ldots, v_d$. Each simplex thus corresponds to an equivalence class of executions that "look the same" to the processes at its vertexes. The term $\mathcal{P}(S^m)$ to denote the subcomplex of $\mathcal{P}$ corresponding to executions in which only the processes in $ids(S^m)$ participate (the rest fail before sending any messages). If $m < n - f$, then there are no such executions, and $\mathcal{P}(S^m)$ is empty. The structure of the protocol complex $\mathcal{P}$ depends both on the protocol and on the timing and failure characteristics of the model. $\mathcal{P}$ often refers to both the protocol and its complex, relying on context to disambiguate.

A protocol *solves* k-set agreement if there is a simplicial map δ, called *decision map*, carrying vertexes of $\mathcal{P}$ to values in V such that if $\vec{p} \in \mathcal{P}(S^n)$ then $\delta(\vec{p}) \in vals(S^n)$, and δ maps the vertexes of any given simplex in $\mathcal{P}(S^n)$ to at most k distinct values.

Applications

The renaming problem is a key tool for understanding the power of various asynchronous models of computation.

Open Problems

Characterizing the full power of the topological approach to proving lower bounds remains an open problem.

Cross References

► Asynchronous Consensus Impossibility
► Renaming

Recommended Reading

Perhaps the first paper to investigate the solvability of distributed tasks was the landmark 1985 paper of Fischer, Lynch, and Paterson [6] which showed that *consensus*, then considered an abstraction of the database commitment problem, had no 1-resilient message-passing solution. Other tasks that attracted attention include *renaming* [1,12,15] and *set agreement* [3,5,12,10,15,17].

In 1988, Biran, Moran, and Zaks [2] gave a graph-theoretic characterization of decision problems that can be solved in the presence of a single failure in a message-passing system. This result was not substantially improved until 1993, when three independent research teams succeeded in applying combinatorial techniques to protocols that tolerate delays by more than one processor: Borowsky and Gafni [3], Saks and Zaharoglou [17], and Herlihy and Shavit [15].

Later, Herlihy and Rajsbaum used homology theory to derive further impossibility results for set agreement and to unify a variety of known impossibility results in terms of the theory of chain maps and chain complexes [12]. Using the same simplicial model.

Biran, Moran, and Zaks [2] gave the first decidability result for decision tasks, showing that tasks are decidable in the 1-resilient message-passing model. Gafni and Koutsoupias [7] were the first to make the important observation that the contractibility problem can be used to prove that tasks are undecidable, and suggest a strategy to reduce a specific wait-free problem for three processes to a contractibility problem. Herlihy and Rajsbaum [11] provide a more extensive collection of decidability results.

Borowsky and Gafni [3], define an iterated immediate snapshot model that has a recursive structure. Chaudhuri, Herlihy, Lynch, and Tuttle [4] give an inductive construction for the synchronous model, and while the resulting "Bermuda Triangle" is visually appealing and an elegant combination of proof techniques from the literature, there is a fair amount of machinery needed in the formal description of the construction. In this sense, the formal presentation of later constructions is substantially more succinct.

More recent work in this area includes separation results [8] and complexity lower bounds [9].

1. Attiya, H., Bar-Noy, A., Dolev, D., Peleg, D., Reischuk, R.: Renaming in an asynchronous environment. J. ACM **37**(3), 524–548 (1990)
2. Biran, O., Moran, S., Zaks, S.: A combinatorial characterization of the distributed 1-solvable tasks. J. Algorithms **11**(3), 420–440 (1990)
3. Borowsky, E., Gafni, E.: Generalized FLP impossibility result for *t*-resilient asynchronous computations. In: Proceedings of the 25th ACM Symposium on Theory of Computing, May 1993
4. Chaudhuri, S., Herlihy, M., Lynch, N.A., Tuttle, M.R.: Tight bounds for k-set agreement. J. ACM **47**(5), 912–943 (2000)
5. Chaudhuri, S.: More choices allow more faults: Set consensus problems in totally asynchronous systems. Inf. Comp. **105**(1), 132–158 (1993) A preliminary version appeared in ACM PODC 1990
6. Fischer, M.J., Lynch, N.A., Paterson, M.S.: Impossibility of distributed consensus with one faulty processor. J. ACM **32**(2), 374–382 (1985)
7. Gafni, E., Koutsoupias, E.: Three-processor tasks are undecidable. SIAM J. Comput. **28**(3), 970–983 (1999)
8. Gafni, E., Rajsbaum, S., Herlihy, M.: Subconsensus tasks: Renaming is weaker than set agreement. In: Lecture Notes in Computer Science, pp. 329–338. (2006)
9. Guerraoui, R., Herlihy, M., Pochon, B.: A topological treatment of early-deciding set-agreement. In: OPODIS, pp. 20–35, (2006)
10. Herlihy, M., Rajsbaum, S.: Set consensus using arbitrary objects. In: Proceedings of the 13th Annual ACM Symposium on Principles of Distributed Computing, pp. 324–333, August (1994)
11. Herlihy, M., Rajsbaum, S.: The decidability of distributed decision tasks (extended abstract). In: STOC '97: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing, pp. 589–598. ACM Press, New York (1997)
12. Herlihy, M., Rajsbaum, S.: Algebraic spans. Math. Struct. Comput. Sci. **10**(4), 549–573 (2000)
13. Herlihy, M., Rajsbaum, S.: A classification of wait-free loop agreement tasks. Theor. Comput. Sci. **291**(1), 55–77 (2003)
14. Herlihy, M., Rajsbaum, S., Tuttle, M.R.: Unifying synchronous and asynchronous message-passing models. In: PODC '98: Proceedings of the seventeenth annual ACM symposium on Principles of distributed computing, pp. 133–142. ACM Press, New York (1998)
15. Herlihy, M., Shavit, N.: The topological structure of asynchronous computability. J. ACM **46**(6), 858–923 (1999)
16. Pease, M., Shostak, R., Lamport, L.: Reaching agreement in the presence of faults. J. ACM **27**(2), 228–234 (1980)
17. Saks, M., Zaharoglou, F.: Wait-free k-set agreement is impossible: The topology of public knowledge. SIAM J. Comput. **29**(5), 1449–1483 (2000)

Trade-Offs for Dynamic Graph Problems
2005; Demetrescu, Italiano

CAMIL DEMETRESCU, GIUSEPPE F. ITALIANO
Department of Computer & Systems Science,
University of Rome, Rome, Italy

Keywords and Synonyms

Trading off update time for query time in dynamic graph problems

Problem Definition

A dynamic graph algorithm maintains a given property $\mathcal{P}$ on a graph subject to dynamic changes, such as edge insertions, edge deletions and edge weight updates. A dynamic graph algorithm should process queries on property $\mathcal{P}$ quickly, and perform update operations faster than recomputing from scratch, as carried out by the fastest static algorithm. A typical definition is given below:

Definition 1 (Dynamic graph algorithm) Given a graph and a graph property $\mathcal{P}$, a *dynamic graph algorithm* is a data structure that supports any intermixed sequence of the following operations:

`insert`(u, v):	insert edge (u, v) into the graph.
`delete`(u, v):	delete edge (u, v) from the graph.
`query`$(\ldots)$:	answer a query about property $\mathcal{P}$ of the graph.

A graph algorithm is *fully dynamic* if it can handle both edge insertions and edge deletions and *partially dynamic* if it can handle either edge insertions or edge deletions, but not both: it is *incremental* if it supports insertions only, and *decremental* if it supports deletions only. Some papers study variants of the problem where more than one edge can be deleted of inserted at the same time, or edge weights can be changed. In some cases, an update may be the insertion or deletion of a node along with all edges incident to them. Some other papers only deal with specific classes of graphs, e. g., planar graphs, directed acyclic graphs (DAGs), etc.

There is a vast literature on dynamic graph algorithms. Graph problems for which efficient dynamic solutions are known include graph connectivity, minimum cut, minimum spanning tree, transitive closure, and shortest paths (see, e. g. [3] and the references therein). Many of them update explicitly the property $\mathcal{P}$ after each update in order to answer queries in optimal time. This may be a good choice in scenarios where there are few updates and many queries. In applications where the numbers of updates and queries are comparable, a better approach would be to try to reduce the update time, possibly at the price of increasing the query time. This is typically achieved by relaxing the assumption that the property $\mathcal{P}$ should be maintained explicitly.

This entry focuses on algorithms for dynamic graph problems that maintain the graph property implicitly, and thus require non-constant query time while supporting faster updates. In particular, it considers two problems: *dynamic transitive closure* (also known as *dynamic reachability*) and *dynamic all-pairs shortest paths*, defined below.

Definition 2 (Fully dynamic transitive closure) The *fully dynamic transitive closure problem* consists of maintaining a directed graph under an intermixed sequence of the following operations:

`insert`(u, v):	insert edge (u, v) into the graph.
`delete`(u, v):	delete edge (u, v) from the graph.
`query`(x, y):	return *true* if there is a directed path from vertex x to vertex y, and *false* otherwise.

Definition 3 (Fully dynamic all-pairs shortest paths) The *fully dynamic transitive closure problem* consists of maintaining a weighted directed graph under an intermixed sequence of the following operations:

`insert`(u, v, w):	insert edge (u, v) into the graph with weight w.
`delete`(u, v):	delete edge (u, v) from the graph.
`query`(x, y):	return the distance from x to y in the graph, or $+\infty$ if there is no directed path from x to y.

Recall that the distance from a vertex x to a vertex y is the weight of a minimum-weight path from x to y, where the weight of a path is defined as the sum of edge weights in the path.

Key Results

This section presents a survey of query/update tradeoffs for dynamic transitive closure and dynamic all-pairs shortest paths.

Dynamic Transitive Closure

The first query/update tradeoff for this problem was devised by Henzinger and King [6], who proved the following result:

Theorem 1 (Henzinger and King 1995 [6]) *Given a general directed graph, there is a randomized algorithm with one-sided error for the fully dynamic transitive closure that supports a worst-case query time of $O(n/\log n)$ and an amortized update time of $O(m\sqrt{n}\log^2 n)$.*

The first subquadratic algorithm for this problem is due to Demetrescu and Italiano for the case of directed acyclic graphs [4,5]:

Theorem 2 (Demetrescu and Italiano 2000 [4,5]) *Given a directed acyclic graph with n vertices, there is a randomized algorithm with one-sided error for the fully dynamic*

Trade-Offs for Dynamic Graph Problems, Table 1
Fully dynamic transitive closure algorithms with implicit solution representation

Type of graphs	Type of algorithm	Update time	Query time	Reference
General	Monte Carlo	$O(m\sqrt{n}\log^2 n)$ amort.	$O(n/\log n)$	HK [6]
DAG	Monte Carlo	$O(n^{1.575})$	$O(n^{0.575})$	DI [4]
General	Monte Carlo	$O(n^{1.575})$	$O(n^{0.575})$	Sank. [13]
General	Monte Carlo	$O(n^{1.495})$	$O(n^{1.495})$	Sank. [13]
General	Deterministic	$O(m\sqrt{n})$ amort.	$O(\sqrt{n})$	RZ [10]
General	Deterministic	$O(m + n\log n)$ amort.	$O(n)$	RZ [11]

transitive closure problem that supports each query in $O(n^{\epsilon})$ time and each insertion/deletion in $O(n^{\omega(1,\epsilon,1)-\epsilon} + n^{1+\epsilon})$, for any $\epsilon \in [0, 1]$, where $\omega(1, \epsilon, 1)$ is the exponent of the multiplication of an $n \times n^{\epsilon}$ matrix by an $n^{\epsilon} \times n$ matrix.

Notice that the dependence of the bounds upon parameter ε leads to a full range of query/update tradeoffs. Balancing the two terms in the update bound of Theorem 2 yields that ε must satisfy the equation $\omega(1, \epsilon, 1) = 1 + 2\epsilon$. The current best bounds on $\omega(1, \epsilon, 1)$ [2,7] imply that $\epsilon < 0.575$. Thus, the smallest update time is $O(n^{1.575})$, which gives a query time of $O(n^{0.575})$:

Corollary 1 (Demetrescu and Italiano 2000 [4,5]) *Given a directed acyclic graph with n vertices, there is a randomized algorithm with one-sided error for the fully dynamic transitive closure problem that supports each query in $O(n^{0.575})$ time and each insertion/deletion in $O(n^{1.575})$ time.*

This result has been generalized to the case of general directed graphs by Sankowski [13]:

Theorem 3 (Sankowsk 2004 [13]) *Given a general directed graph with n vertices, there is a randomized algorithm with one-sided error for the fully dynamic transitive closure problem that supports each query in $O(n)$ time and each insertion/deletion in $O(n^{\omega(1,\epsilon,1)-\epsilon} + n^{1+\epsilon})$, for any $\epsilon \in [0, 1]$, where $\omega(1, \epsilon, 1)$ is the exponent of the multiplication of an $n \times n^{\epsilon}$ matrix by an $n^{\epsilon} \times n$ matrix.*

Corollary 2 (Sankowski 2004 [13]) *Given a general directed graph with n vertices, there is a randomized algorithm with one-sided error for the fully dynamic transitive closure problem that supports each query in $O(n^{0.575})$ time and each insertion/deletion in $O(n^{1.575})$ time.*

Sankowski has also shown how to achieve an even faster update time of $O(n^{1.495})$ at the expense of a much higher $O(n^{1.495})$ query time:

Theorem 4 (Sankowski 2004 [13]) *Given a general directed graph with n vertices, there is a randomized algorithm with one-sided error for the fully dynamic transitive closure problem that supports each query and each insertion/deletion in $O(n^{1.495})$ time.*

Roditty and Zwick presented algorithms designed to achieve better bounds in the case of sparse graphs:

Theorem 5 (Roditty and Zwick 2002 [10]) *Given a general directed graph with n vertices and m edges, there is a deterministic algorithm for the fully dynamic transitive closure problem that supports each insertion/deletion in $O(m\sqrt{n})$ amortized time and each query in $O(\sqrt{n})$ worst-case time.*

Theorem 6 (Roditty and Zwick 2004 [11]) *Given a general directed graph with n vertices and m edges, there is a deterministic algorithm for the fully dynamic transitive closure problem that supports each insertion/deletion in $O(m + n\log n)$ amortized time and each query in $O(n)$ worst-case time.*

Observe that the results of Theorem 5 and Theorem 6 are subquadratic for $m = o(n^{1.5})$ and $m = o(n^2)$, respectively. Moreover, they are not based on fast matrix multiplication, which is theoretically efficient but impractical.

Dynamic Shortest Paths

The first effective tradeoff algorithm for dynamic shortest paths is due to Roditty and Zwick in the special case of sparse graphs with unit edge weights [12]:

Theorem 7 (Roditty and Zwick 2004 [12]) *Given a general directed graph with n vertices, m edges, and unit edge weights, there is a randomized algorithm with one-sided error for the fully dynamic all-pairs shortest paths problem that supports each distance query in $O(t + \frac{n\log n}{k})$ worst-case time and each insertion/deletion in $O(\frac{mn^2\log n}{t^2} + km + \frac{mn\log n}{k})$ amortized time.*

By choosing $k = (n\log n)^{1/2}$ and $(n\log n)^{1/2} \le t \le n^{3/4}(\log n)^{1/4}$ in Theorem 7, it is possible to obtain an amortized update time of $O(\frac{mn^2\log n}{t^2})$ and a worst-case query time of $O(t)$. The fastest update time of $O(m\sqrt{n \log n})$ is obtained by choosing $t = n^{3/4}(\log n)^{1/4}$.

Later, Sankowski devised the first subquadratic algorithm for dense graphs based on fast matrix multiplication [14]:

Theorem 8 (Sankowski 2005 [14]) *Given a general directed graph with n vertices and unit edge weights, there is a randomized algorithm with one-sided error for the fully dynamic all-pairs shortest paths problem that supports each distance query in* $O(n^{1.288})$ *time and each insertion/deletion in* $O(n^{1.932})$ *time.*

Applications

The transitive closure problem studied in this entry is particularly relevant to the field of databases for supporting transitivity queries on dynamic graphs of relations [16]. The problem also arises in many other areas such as compilers, interactive verification systems, garbage collection, and industrial robotics.

Application scenarios of dynamic shortest paths include network optimization [1], document formatting [8], routing in communication systems, robotics, incremental compilation, traffic information systems [15], and dataflow analysis. A comprehensive review of real-world applications of dynamic shortest path problems appears in [9].

Open Problems

It is a fundamental open problem whether the fully dynamic all pairs shortest paths problem of Definition 3 can be solved in subquadratic time per operation in the case of graphs with real-valued edge weights.

Cross References

- ▶ All Pairs Shortest Paths in Sparse Graphs
- ▶ All Pairs Shortest Paths via Matrix Multiplication
- ▶ Decremental All-Pairs Shortest Paths
- ▶ Fully Dynamic All Pairs Shortest Paths
- ▶ Fully Dynamic Transitive Closure
- ▶ Single-Source Fully Dynamic Reachability

Recommended Reading

1. Ahuja, R., Magnanti, T., Orlin, J.: Network Flows: Theory, Algorithms and Applications. Prentice Hall, Englewood Cliffs (1993)
2. Coppersmith, D., Winograd, S.: Matrix multiplication via arithmetic progressions. J. Symb. Comput. **9**, 251–280 (1990)
3. Demetrescu, C., Finocchi, I., Italiano, G.: Dynamic Graphs. In: Mehta, D., Sahni, S. (eds.) Handbook on Data Structures and Applications (CRC Press Series, in Computer and Information Science), chap. 36. CRC Press, Boca Raton (2005)
4. Demetrescu, C., Italiano, G.: Fully dynamic transitive closure: Breaking through the $O(n^2)$ barrier. In: Proc. of the 41st IEEE Annual Symposium on Foundations of Computer Science (FOCS'00), Redondo Beach (2000), pp. 381–389
5. Demetrescu, C., Italiano, G.: Trade-offs for fully dynamic reachability on dags: Breaking through the $O(n^2)$ barrier. J. ACM **52**, 147–156 (2005)
6. Henzinger, M., King, V.: Fully dynamic biconnectivity and transitive closure. In: Proc. 36th IEEE Symposium on Foundations of Computer Science (FOCS'95), Milwaukee (1995), pp. 664–672
7. Huang, X., Pan, V.: Fast rectangular matrix multiplication and applications. J. Complex. **14**, 257–299 (1998)
8. Knuth, D., Plass, M.: Breaking paragraphs into lines. Software-practice Exp. **11**, 1119–1184 (1981)
9. Ramalingam, G.: Bounded incremental computation. In: Lecture Notes in Computer Science, vol. 1089. Springer, New York (1996)
10. Roditty, L., Zwick, U.: Improved dynamic reachability algorithms for directed graphs. In: Proceedings of 43th Annual IEEE Symposium on Foundations of Computer Science (FOCS), Vancouver (2002), pp. 679–688
11. Roditty, L., Zwick, U.: A fully dynamic reachability algorithm for directed graphs with an almost linear update time. In: Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC), Chicago (2004), pp. 184–191
12. Roditty, L., Zwick, U.: On Dynamic Shortest Paths Problems. In: Proceedings of the 12th Annual European Symposium on Algorithms (ESA), Bergen (2004), pp. 580–591
13. Sankowski, P.: Dynamic transitive closure via dynamic matrix inverse. In: FOCS '04: Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS'04), pp. 509–517. IEEE Computer Society, Washington, DC (2004)
14. Sankowski, P.: Subquadratic algorithm for dynamic shortest distances. In: 11th Annual International Conference on Computing and Combinatorics (COCOON'05), Kunming (2005), pp. 461–470
15. Schulz, F., Wagner, D., Weihe, K.: Dijkstra's algorithm on-line: an empirical case study from public railroad transport. In: Proc. 3rd Workshop on Algorithm Engineering (WAE'99), London (1999), pp. 110–123
16. Yannakakis, M.: Graph-theoretic methods in database theory. In: Proc. 9-th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Nashville (1990). pp. 230–242

Traveling Sales Person with Few Inner Points

2004; Deĭneko, Hoffmann, Okamoto, Woeginger

Yoshio Okamoto
Department of Information and Computer Sciences, Toyohashi University of Technology, Toyohashi, Japan

Keywords and Synonyms

Traveling salesman problem; Traveling salesperson problem; Minimum-cost Hamiltonian circuit problem;

Minimum-weight Hamiltonian circuit problem; Minimum-cost Hamiltonian cycle problem; Minimum-weight Hamiltonian cycle problem

Problem Definition

In the *traveling salesman problem* (TSP) n cities $1, 2, \ldots, n$ together with all the pairwise distances $d(i,j)$ between cities i and j are given. The goal is to find the shortest tour that visits every city exactly once and in the end returns to its starting city. The TSP is one of the most famous problems in combinatorial optimization, and it is well-known to be NP-hard. For more information on the TSP, the reader is referred to the book by Lawler, Lenstra, Rinnooy Kan, and Shmoys [14].

A special case of the TSP is the so-called *Euclidean TSP*, where the cities are points in the Euclidean plane, and the distances are simply the Euclidean distances. A special case of the Euclidean TSP is the *convex Euclidean TSP*, where the cities are further restricted so that they lie in convex position. The Euclidean TSP is still NP-hard [4,17], but the convex Euclidean TSP is quite easy to solve: Running along the boundary of the convex hull yields a shortest tour. Motivated by these two facts, the following natural question is posed: What is the influence of the number of inner points on the complexity of the problem? Here, an *inner point* of a finite point set P is a point from P which lies in the interior of the convex hull of P. Intuition says that "Fewer inner points make the problem easier to solve."

The result below answers this question and supports the intuition above by providing simple exact algorithms.

Key Results

Theorem 1 *The special case of the Euclidean TSP with few inner points can be solved in the following time and space complexity. Here, n denotes the total number of cities and k denotes the number of cities in the interior of the convex hull. 1. In time* $O(k!kn)$ *and space* $O(k)$*. 2. In time* $O(2^k k^2 n)$ *and space* $O(2^k kn)$ *[1].*

Here, assume that the convex hull of a given point set is already determined, which can be done in time $O(n \log n)$ and space $O(n)$. Further, note that the above space bounds do not count the space needed to store the input but they just count the space in working memory (as usual in theoretical computer science).

Theorem 1 implies that, from the viewpoint of parameterized complexity [2,3,16], these algorithms are fixed-parameter algorithms, when the number k of inner points is taken as a parameter, and hence the problem is fixed-parameter tractable (FPT). (A *fixed-parameter algorithm* has running time $O(f(k)\text{poly}(n))$, where n is the input size, k is a parameter and $f: \mathbb{N} \to \mathbb{N}$ is an arbitrary computable function. For example, an algorithm with running time $O(5^k n)$ is a fixed-parameter algorithm whereas one with $O(n^k)$ is not.) Observe that the second algorithm gives a polynomial-time exact solution to the problem when $k = O(\log n)$.

The method can be extended to some generalized versions of the TSP. For example, Deĭneko et al. [1] stated that the prize-collecting TSP and the partial TSP can be solved in a similar manner.

Applications

The theorem is motivated more from a theoretical side rather than an application side. No real-world application has been assumed.

As for the theoretical application, the viewpoint (introduced in the problem definition section) has been applied to other geometric problems. Some of them are listed below.

The Minimum Weight Triangulation Problem: Given n points in the Euclidean plane, the problem asks to find a triangulation of the points which has minimum total length. The problem is now known to be NP-hard [15].

Hoffmann and Okamoto [10] proved that the problem is fixed-parameter tractable with respect to the number k of inner points. The time complexity they gave is $O(6^k n^5 \log n)$. This is subsequently improved by Grantson, Borgelt, and Levcopoulos [6] to $O(4^k kn^4)$ and by Spillner [18] to $O(2^k kn^3)$. Yet other fixed-parameter algorithms have also been proposed by Grantson, Borgelt, and Levcopoulos [7,8]. The currently best time complexity was given by Knauer and Spillner [13] and it is $O(2^{c\sqrt{k}\log k} k^{3/2} n^3)$ where $c = (2+\sqrt{2})/(\sqrt{3}-\sqrt{2}) < 11$.

The Minimum Convex Partition Problem: Given n points in the Euclidean plane, the problem asks to find a partition of the convex hull of the points into the minimum number of convex regions having some of the points as vertices.

Grantson and Levcopoulos [9] gave an algorithm running in $O(k^{6k-5} 2^{16k} n)$ time. Later, Spillner [19] improved the time complexity to $O(2^k k^3 n^3)$.

The Minimum Weight Convex Partition Problem: Given n points in the Euclidean plane, the problem asks to find a convex partition of the points with minimum total length.

Grantson [5] gave an algorithm running in $O(k^{6k-5}2^{16k}n)$ time. Later, Spillner [19] improved the time complexity to $O(2^k k^3 n^3)$.

The Crossing Free Spanning Tree Problem: Given an n-vertex geometric graph (i. e., a graph drawn on the Euclidean plane where every edge is a straight line segment connecting two distinct points), the problem asks to determine whether it has a spanning tree without any crossing of the edges. Jansen and Woeginger [11] proved this problem is NP-hard.

Knauer and Spillner [12] gave algorithms running in $O(175^k k^2 n^3)$ time and $O(2^{33\sqrt{k}\log k} k^2 n^3)$ time.

The method proposed by Knauer and Spillner [12] can be adopted to the TSP as well. According to their result, the currently best time complexity for the TSP is $2^{O(\sqrt{k}\log k)}\text{poly}(n)$.

Open Problems

Currently, no lower bound result for the time complexity seems to be known. For example, is it possible to prove under a reasonable complexity-theoretic assumption the impossibility for the existence of an algorithm running in $2^{O(\sqrt{k})}\text{poly}(n)$ for the TSP?

Cross References

On the traveling salesman problem:
▶ Euclidean Traveling Salesperson Problem
▶ Hamilton Cycles in Random Intersection Graphs
▶ Implementation Challenge for TSP Heuristics
▶ Metric TSP
On fixed-parameter algorithms:
▶ Closest Substring
▶ Parameterized SAT
▶ Vertex Cover Kernelization
▶ Vertex Cover Search Trees
On others:
▶ Minimum Weight Triangulation

Recommended Reading

1. Deĭneko, V.G., Hoffmann, M., Okamoto, Y., Woeginger, G.J.: The traveling salesman problem with few inner points. Oper. Res. Lett. **31**, 106–110 (2006)
2. Downey, R.G., Fellows, M.R.: Parameterized Complexity. In: Monographs in Computer Science. Springer, New York (1999)
3. Flum, J., Grohe, M.: Parameterized Complexity Theory. Texts in Theoretical Computer Science An EATCS Series. Springer, Berlin (2006)
4. Garey, M.R., Graham, R.L., Johnson, D.S.: Some NP-complete geometric problems. In: Proceedings of 8th Annual ACM Symposium on Theory of Computing (STOC '76), pp. 10–22. Association for Computing Machinery, New York (1976)
5. Grantson, M.: Fixed-parameter algorithms and other results for optimal partitions. Lecentiate Thesis, Department of Computer Science, Lund University (2004)
6. Grantson, M., Borgelt, C., Levcopoulos, C.: A fixed parameter algorithm for minimum weight triangulation: Analysis and experiments. Tech. Rep. 154, Department of Computer Science, Lund University (2005)
7. Grantson, M., Borgelt, C., Levcopoulos, C.: Minimum weight triangulation by cutting out triangles. In: Deng, X., Du, D.-Z. (eds.) Proceedings of the 16th Annual International Symposium on Algorithms and Computation (ISAAC). Lecture Notes in Computer Science, vol. 3827, pp. 984–994. Springer, New York (2005)
8. Grantson, M., Borgelt, C., Levcopoulos, C.: Fixed parameter algorithms for the minimum weight triangulation problem. Tech. Rep. 158, Department of Computer Science, Lund University (2006)
9. Grantson, M., Levcopoulos, C.: A fixed parameter algorithm for the minimum number convex partition problem. In: Akiyama, J., Kano, M., Tan, X. (eds.) Proceedings of Japanese Conference on Discrete and Computational Geometry (JCDCG 2004). Lecture Notes in Computer Science, vol. 3742, pp. 83–94. Springer, New York (2005)
10. Hoffmann, M., Okamoto, Y.: The minimum weight triangulation problem with few inner points. Comput. Geom. Theory Appl. **34**, 149–158 (2006)
11. Jansen, K., Woeginger, G.J.: The complexity of detecting crossingfree configurations in the plane. BIT **33**, 580–595 (1993)
12. Knauer, C., Spillner, A.: Fixed-parameter algorithms for finding crossing-free spanning trees in geometric graphs. Tech. Rep. 06–07, Department of Computer Science, Friedrich-Schiller-Universität Jena (2006)
13. Knauer, C., Spillner, A.: A fixed-parameter algorithm for the minimum weight triangulation problem based on small graph separators. In: Proceedings of the 32nd International Workshop on Graph-Theoretic Concepts in Computer Science (WG). Lecture Notes in Computer Science, vol. 4271, pp. 49–57. Springer, New York (2006)
14. Lawler, E., Lenstra, J., Rinnooy Kan, A., Shmoys, D. (eds.): The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization. Wiley, Chichester (1985)
15. Mulzer, W., Rote, G.: Minimum Weight Triangulation is NP-hard. In: Proceedings of the 22nd Annual ACM Symposium on Computational Geometry (SoCG), Association for Computing Machinery, New York 2006, pp. 1–10
16. Niedermeier, R.: Invitation to Fixed-Parameter Algorithms. Oxford Lecture Series in Mathematics and Its Applications, vol. 31. Oxford University Press, Oxford (2006)
17. Papadimitriou, C.H.: The Euclidean travelling salesman problem is NP-complete. Theor. Comput. Sci. **4**, 237–244 (1977)
18. Spillner, A.: A faster algorithm for the minimum weight triangulation problem with few inner points. In: Broersma, H., Johnson, H., Szeider, S. (eds.) Proceedings of the 1st ACiD Workshop. Texts in Algorithmics, vol. 4, pp. 135–146. King's College, London (2005)
19. Spillner, A.: Optimal convex partitions of point sets with few inner points. In: Proceedings of the 17th Canadian Conference on Computational Geometry (CCCG), 2005, pp. 34–37

Traveling Salesperson Problem

▶ Traveling Sales Person with Few Inner Points

Tree Agreement

▶ Maximum Agreement Subtree (of 2 Binary Trees)

Tree Alignment

▶ Maximum Agreement Subtree (of 3 or More Trees)

Tree Compression and Indexing

2005; Ferragina, Luccio, Manzini, Muthukrishnan

PAOLO FERRAGINA[1], S. SRINIVASA RAO[2]
[1] Department of Computer Science, University of Pisa, Pisa, Italy
[2] Computational Logic and Algorithms Group, IT University of Copenhagen, Copenhagen, Denmark

Keywords and Synonyms

XML compression and indexing

Problem Definition

Trees are a fundamental structure in computing. They are used in almost every aspect of modeling and representation for explicit computation like searching for keys, maintaining directories, and representations of parsing or execution traces—to name just a few. One of the latest uses of trees is XML, the de facto format for data storage, integration, and exchange over the Internet (see http://www.w3.org/XML/). Explicit storage of trees, with one pointer per child as well as other auxiliary information (e. g. label), is often taken as given but can account for the dominant storage cost. Just to have an idea, a simple tree encoding needs at least 16 bytes per tree node: one pointer to the auxiliary information (e. g. node label) plus three node pointers to the parent, the first child, and the next sibling. This large space occupancy may even prevent the processing of medium size trees, e. g. XML documents. This entry surveys the best known storage solutions for unlabeled and labeled trees that are space efficient and support fast navigational and search operations over the tree structure. In the literature, they are referred to as *succinct/compressed tree indexing* solutions.

Notation and Basic Facts

The information-theoretic storage cost for any item of a universe U can be derived via a simple *counting argument*: at least $\log|U|$ bits are needed to distinguish any two items of U.[1] Now, let $\mathcal{T}$ be a rooted tree of arbitrary degree and shape, and consider the following three main classes of trees:

Ordinal Trees. $\mathcal{T}$ is unlabeled and its children are left-to-right *ordered*. The number of ordinal trees on t nodes is $C_t = \binom{c2t}{t}/(t+1)$ which induces a lower bound of $2t - \Theta(\log t)$ bits.

Cardinal k-ary Trees $\mathcal{T}$ is labeled on its *edges* with symbols drawn from the alphabet $\Sigma = \{1, \ldots, k\}$. Any node has degree at most k because the edges outgoing from each node have *distinct* labels. Typical examples of cardinal trees are the binary tree ($k = 2$), the (uncompacted) tree and the *Patricia tree*. The number of k-ary cardinal trees on t nodes is $C_t^k = \binom{kt+1}{t}/(kt+1)$ which induces a lower bound of $t(\log k + \log e)$ bits, when k is a slowly-growing function of t.

(Multi-)Labeled Trees. $\mathcal{T}$ is an ordinal tree, labeled on its *nodes* with symbols drawn from the alphabet Σ. In the case of multi-labeled trees, every node has *at least* one symbol as its label. The *same* symbols may repeat among sibling nodes, so that the degree of each node is *unbounded*, and the same labeled-subpath may occur many times in $\mathcal{T}$, anchored anywhere. The information-theoretic lower bound on the storage complexity of this class of trees on t nodes comes easily from the decoupling of the tree structure and the storage of tree labels. For labeled trees it is $\log C_t + t\log|\Sigma| = t(\log|\Sigma| + 2) - \Theta(\log t)$ bits.

The following query operations should be supported over $\mathcal{T}$:

Basic Navigational Queries. They ask for the parent of node u, the ith child of u, the degree of u. These operations may be restricted to some label $c \in \Sigma$, if $\mathcal{T}$ is labeled.

Sophisticated Navigational Queries. They ask for the jth level-ancestor of u, the depth of u, the subtree size of u, the lowest common ancestor of a pair of nodes, the ith node according to some node ordering over $\mathcal{T}$, possibly restricted to some label $c \in \Sigma$ (if $\mathcal{T}$ is labeled). For even more operations see [2,11].

[1]Throughout the entry, all logarithms are taken to the base 2, and it is assumed $0 \log 0 = 0$.

Subpath Query. Given a labeled subpath Π, it asks for the (number *occ* of) nodes of $\mathcal{T}$ that immediately descend from Π. Every subpath occurrence may be anchored anywhere in the tree (i. e. not necessarily in its root).

The elementary solution to the tree indexing problem consists of encoding the tree $\mathcal{T}$ via a mixture of pointers and arrays, thus taking a total of $\Theta(t \log t)$ bits. This supports basic navigational queries in constant time, but it is not space efficient and requires the whole visit of the tree to implement the subpath query or the more sophisticated navigational queries. Here the goal is to design tree storage schemes that are either *succinct*, namely "close to the information-theoretic lower bound" mentioned before, or *compressed* in that they achieve "entropy-bounded storage." Furthermore, these storage schemes do *not* require the whole visit of the tree for most navigational operations. Thus, succinct/compressed tree indexing solutions are distinct from simply compressing the input, and then uncompressing it later on at query time.

In this entry, it is assumed that $t \geq |\Sigma|$ and the Random Access Machine (RAM) with word size $\Theta(\lg t)$ is taken as the model of computation. This way, one can perform various arithmetic and bit-wise boolean operations on single words in constant time.

Key Results

The notion of *succinct* data structures was introduced by Jacobson [10] in a seminal work over 18 years ago. He presented a storage scheme for ordinal trees using $2t + o(t)$ bits and supporting basic navigational queries in $O(\log \log t)$ time (i. e. parent, first child and next sibling of a node). Later, Munro and Raman [13] closed the issue for ordinal trees on basic navigational queries and the subtree-size query by achieving constant query-time and $2t + o(t)$ bits of storage. Their storage scheme is called *Balanced Parenthesis* (BP).[2] Subsequently, Benoit et al. [3] proposed a storage scheme called *Depth-First Unary Degree Sequence* (shortly, DFUDS) that still uses $2t + o(t)$ bits but performs more navigational queries like ith child, child rank, and node degree in constant time. Geary et al. [8] gave another representation still taking optimal space that extends DFUDS's operations to the level-ancestor query.

Although these three representations achieve the optimal space occupancy, none of them supports every existing operation in constant time: e. g. BP does not support ith child and child rank, DFUDS and Geary et al.'s representation do not support LCA. Recently, Jansson et al. [11] extended the DFUDS storage scheme in two directions: (1) they showed how to implement in constant time all navigational queries above;[3] (2) they showed how to compress the new tree storage scheme up to $H^*(\mathcal{T})$, which denotes the entropy of the distribution of node degrees in $\mathcal{T}$.

Theorem 1 ([Jansson et al. 2007]) *For any rooted tree $\mathcal{T}$ with t nodes, there exists a tree indexing scheme that uses $tH^*(\mathcal{T}) + O(t(\log \log t)^2/\log t)$ bits and supports all navigational queries in constant time.*

This improves the standard tree pointer-based representation, since it needs no more than $H^*(\mathcal{T})$ bits per node and does not compromise the performance of sophisticated navigational queries. Since it is $H^*(\mathcal{T}) \leq 2$, this solution is also never worse than BP or DFUDS, but its improvement may be significant! This result can be extended to achieve the kth order entropy of the DFUDS sequence, by adopting any compressed-storage scheme for strings (see e. g. [7] and references therein).

Benoit et al. [3] extended the use of DFUDS to cardinal trees, and proposed a tree indexing scheme whose space occupancy is close to the information-theoretic lower bound and supports various navigational queries in constant time. Raman et al. [15] improved the space by using a different approach (based on storing the tree as a set of edges) thus proving the following:

Theorem 2 ([Raman et al. 2002]) *For any k-ary cardinal tree $\mathcal{T}$ with t nodes, there exists a tree indexing scheme that uses $\log C_t^k + o(t) + O(\log \log k)$ bits and supports in constant time the following operations: finding the parent, the degree, the ordinal position among its siblings, the child with label c, the ith child of a node.*

The subtree size operation cannot be supported efficiently using this representation, so [3] should be resorted to in case this operation is a primary concern.

Despite this flurry of activity, the fundamental problem of indexing *labeled* trees succinctly has remained mostly unsolved. In fact, the succinct encoding for ordered trees mentioned above might be replicated $|\Sigma|$ times (one per possible symbol of Σ), and then the divide-and-conquer approach of [8] might be applied to reduce the final space occupancy. However, the final space bound

[2] Some papers [Chiang et al., ACM-SIAM SODA '01; Sadakane, ISAAC '01; Munro et al., J.ALG '01; Munro and Rao, ICALP '04] have extended BP to support in constant time other sophisticated navigational queries like LCA, node degree, rank/select on leaves and number of leaves in a subtree, level-ancestor and level-successor.

[3] The BP representation and the one of Geary et al. [8] have been recently extended to support further operations—like depth/height of a node, next node in the same level, rank/select over various node orders—still in constant time and $2t + o(t)$ bits (see [9] and references therein).

would be $2t + t \log|\Sigma| + O(t|\Sigma|(\log\log\log t)/(\log\log t))$ bits, which is nonetheless far from the information-theoretic storage bound even for moderately large Σ. On the other hand, if subpath queries are of primary concern (e. g. XML), one can use the approach of [12] which consists of a variant of the suffix-tree data structure properly designed to index all $\mathcal{T}$'s labeled paths. Subpath queries can be supported in $O(|\Pi| \log|\Sigma| + occ)$ time, but the required space would be still $\Theta(t \log t)$ bits (with large hidden constants due to the use of suffix trees). Recently, some papers [1,2,5] addressed this problem in its whole generality by either dealing simultaneously with subpath and basic navigational queries [5], or by considering *multi*-labeled trees and a larger set of navigational operations [1,2].

The tree-indexing scheme of [5] is based on a *transform* of the labeled tree $\mathcal{T}$, denoted $\mathsf{xbw}[\mathcal{T}]$, which linearizes it into *two* coordinated arrays $\langle S_{\mathsf{last}}, S_\alpha \rangle$: the former capturing the tree structure and the latter keeping a permutation of the labels of $\mathcal{T}$. $\mathsf{xbw}[\mathcal{T}]$ has the optimal (up to lower-order terms) size of $2t + t\log|\Sigma|$ bits and can be built and inverted in optimal linear time. In designing the XBW-Transform, the authors were inspired by the elegant Burrows–Wheeler transform for strings [4]. The power of $\mathsf{xbw}[\mathcal{T}]$ relies on the fact that it allows one to transform compression and indexing problems on labeled trees into easier problems over strings. Namely, the following two string-search primitives are key tools for indexing $\mathsf{xbw}[\mathcal{T}]$: $\mathsf{rank}_c(S, i)$ returns the number of occurrences of the symbol c in the string prefix $S[1, i]$, and $\mathsf{select}_c(S, j)$ returns the position of the jth occurrence of the symbol c in string S. The literature offers many time/space efficient solutions for these primitives that could be used as a *black-box* for the compressed indexing of $\mathsf{xbw}[\mathcal{T}]$ (see e. g. [2,14] and references therein).

Theorem 3 ([Ferragina et al. 2005]) *Consider a tree $\mathcal{T}$ consisting of t nodes labeled with symbols drawn from alphabet Σ. There exists a compressed tree-indexing scheme that achieves the following performance:*

- *If $|\Sigma| = O(\text{polylog}(t))$, the index takes at most $tH_0(S_\alpha) + 2t + o(t)$ bits, supports basic navigational queries in constant time and (counting) subpath queries in $O(|\Pi|)$ time.*
- *For any alphabet Σ, the index takes less than $tH_k(S_\alpha) + 2t + o(t\log|\Sigma|)$ bits, but label-based navigational queries and (counting) subpath queries are slowed down by a factor $o((\log\log|\Sigma|)3)$.*

Here $H_k(s)$ is the kth order empirical entropy of string s, with $H_k(s) \le H_{k-1}(s)$ for any $k > 0$.

Since $H_k(S_\alpha) \le H_0(S_\alpha) \le \log|\Sigma|$, the indexing of $\mathsf{xbw}[\mathcal{T}]$ takes at most as much space as its plain representation, up to lower order terms, but with the additional feature of being able to navigate and search $\mathcal{T}$ efficiently. This is indeed a sort of *pointerless representation* of the labeled tree $\mathcal{T}$ with additional search functionalities (see [5] for details).

If sophisticated navigational queries over labeled trees are a primary concern, and subpath queries are not necessary, then the approach of Barbay et al. [1,2] should be followed. They proposed the novel concept of *succinct index*, which is different from the concept of *succinct/compressed encoding* implemented by all the above solutions. A succinct index does not *touch* the data to be indexed, it just accesses the data via basic operations offered by the underlying abstract data type (ADT), and requires asymptotically less space than the information-theoretic lower bound on the storage of the data itself. The authors reduce the problem of indexing labeled trees to the one of indexing ordinal trees and strings; and the problem of indexing multi-labeled trees to the one of indexing ordinal trees and binary relations. Then, they provide succinct indexes for strings and binary relations. In order to present their result, the following definitions are needed. Let m be the total number of symbols in $\mathcal{T}$, t_c be the number of nodes labeled c in $\mathcal{T}$, and let ρ_c be the maximum number of labels c in any rooted path of $\mathcal{T}$ (called the *recursivity* of c). Define ρ as the average recursivity of $\mathcal{T}$, namely $\rho = (1/m)\sum_{c\in\Sigma}(t_c\rho_c)$.

Theorem 4 ([Barbay et al. 2007]) *Consider a tree $\mathcal{T}$ consisting of t nodes (multi-)labeled with possibly many symbols drawn from alphabet Σ. Let m be the total number of symbols in $\mathcal{T}$, and assume that the underlying ADT for $\mathcal{T}$ offers basic navigational queries in constant time and retrieves the ith label of a node in time f. There is a succinct index for $\mathcal{T}$ using $m(\log\rho + o(\log(|\Sigma|\rho)))$ bits that supports for a given node u the following operations (where $L = \log\log|\Sigma| \, \log\log\log|\Sigma|$):*

- *Every c-descendant or c-child of u can be retrieved in $O(L(f + \log\log|\Sigma|))$ time.*
- *The set A of c-ancestors of u can be retrieved in $O(L(f + \log\log|\Sigma|) + |A|(\log\log\rho_c + \log\log\log|\Sigma|(f + \log\log|\Sigma|)))$ time.*

Applications

As trees are ubiquitous in many applications, this section concentrates just on two examples that, in their simplicity, highlight the flexibility and power of succinct/compressed tree indexes.

The first example regards suffix trees, which are a crucial algorithmic block of many string processing applications—ranging from bioinformatics to data mining, from data compression to search engines. Standard implementations of suffix trees take at least 80 bits per node. The compressed suffix tree of a string $S[1, s]$ consists of three components: the tree topology, the string depths stored into the internal suffix-tree nodes, and the suffix pointers stored in the suffix-tree leaves (also called *suffix array* of S). The succinct tree representation of [11] can be used to encode the suffix-tree topology and the string depths taking $4s + o(s)$ bits (assuming w.l.o.g. that $|\Sigma| = 2$). The suffix array can be compressed up to the kth order entropy of S via any solution surveyed in [14]. The overall result is never worse than 80 bits per node, but can be significantly better for highly compressible strings.

The second example refers to the XML format which is often modeled as a labeled tree. The succinct/compressed indexes in [1,2,5] are theoretical in flavor but turn out to be relevant for practical XML processing systems. As an example, [6] has published some initial encouraging experimental results that highlight the impact of the XBW-Transform on real XML datasets. The authors show that a proper adaptation of the XBW-Transform allows one to compress XML data up to state-of-the-art XML-conscious compressors, and to provide access to its content, navigate up and down the XML tree structure, and search for simple path expressions and substrings in a few milliseconds over MBs of XML data, by uncompressing only a tiny fraction of them at each operation. Previous solutions took several seconds per operation!

Open Problems

For a complete set of open problems and further directions of research, the interested reader is referred to the recommended readings. Here two main problems, which naturally derive from the discussion above, are commented.

Motivated by XML applications, one may like to extend the subpath search operation to the *efficient* search for all leaves of $\mathcal{T}$ whose labels *contain* a substring β and that *descend from* a given subpath Π. The term "efficient" here means in time proportional to $|\Pi|$ and to the number of retrieved occurrences, but independent as much as possible of $\mathcal{T}$'s size in the worst case. Currently, this search operation is possible only for the leaves which are immediate descendants of Π, and even for this setting, the solution proposed in [6] is not optimal.

There are two main encodings for trees which lead to the results above: ordinal tree representation (BP, DFUDS or the representation of Geary et al. [8]) and XBW. The former is at the base of solutions for sophisticated navigational operations, and the latter is at the base of solutions for sophisticated subpath searches. Is it possible to devise *one unique* transform for the labeled tree $\mathcal{T}$ which combines the best of the two worlds and is still compressible?

Experimental Results

See http://cs.fit.edu/~mmahoney/compression/text.html and at the paper [6] for numerous experiments on XML datasets.

Data Sets

See http://cs.fit.edu/~mmahoney/compression/text.html and the references in [6].

URL to Code

Paper [6] contains a list of software tools for compression and indexing of XML data.

Cross References

- ▶ Compressed Text Indexing
- ▶ Rank and Select Operations on Binary Strings
- ▶ Succinct Encoding of Permutations: Applications to Text Indexing
- ▶ Table Compression
- ▶ Text Indexing

Recommended Reading

1. Barbay, J., Golynski, A., Munro, J.I., Rao, S.S.: Adaptive searching in succinctly encoded binary relations and tree-structured documents. In: Proc. 17th Combinatorial Pattern Matching (CPM). LNCS n. 4009 Springer, Barcelona (2006), pp. 24–35
2. Barbay, J., He, M., Munro, J.I., Rao, S.S.: Succinct indexes for strings, binary relations and multi-labeled trees. In: Proc. 18th ACM-SIAM Symposium on Discrete Algorithms (SODA), New Orleans, USA, (2007), pp. 680–689
3. Benoit, D., Demaine, E., Munro, J.I., Raman, R., Raman, V., Rao, S.S.: Representing trees of higher degree. Algorithmica **43**, 275–292 (2005)
4. Burrows, M., Wheeler, D.: A block sorting lossless data compression algorithm. Tech. Report 124, Digital Equipment Corporation (1994)
5. Ferragina, P., Luccio, F., Manzini, G., Muthukrishnan, S.: Structuring labeled trees for optimal succinctness, and beyond. In: Proc. 46th IEEE Symposium on Foundations of Computer Science (FOCS), pp. 184–193. Cambridge, USA (2005)
6. Ferragina, P., Luccio, F., Manzini, G., Muthukrishnan, S.: Compressing and searching XML data via two zips. In: Proc. 15th World Wide Web Conference (WWW), pp. 751–760. Edingburg, UK(2006)

7. Ferragina, P., Venturini, R.: A simple storage scheme for strings achieving entropy bounds. Theor. Comput. Sci. **372**, (1):115–121 (2007)
8. Geary, R., Raman, R., Raman, V.: Succinct ordinal trees with level-ancestor queries. In: Proc. 15th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 1–10. New Orleans, USA (2004)
9. He, M., Munro, J.I., Rao, S.S.: Succinct ordinal trees based on tree covering. In: Proc. 34th International Colloquium on Algorithms, Language and Programming (ICALP). LNCS n. 4596, pp. 509–520. Springer, Wroclaw, Poland (2007)
10. Jacobson, G.: Space-efficient static trees and graphs. In: Proc. 30th IEEE Symposium on Foundations of Computer Science (FOCS), pp. 549–554. Triangle Park, USA (1989)
11. Jansson, J., Sadakane, K., Sung, W.: Ultra-succinct representation of ordered trees. In: Proc. 18th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 575–584. New Orleans, USA(2007)
12. Kosaraju, S.R.: Efficient tree pattern matching. In: Proc. 20th IEEE Foundations of Computer Science (FOCS), pp. 178–183. Triangle Park, USA (1989)
13. Munro, J.I., Raman, V.: Succinct representation of balanced parentheses and static trees. SIAM J. Comput. **31**(3), 762–776 (2001)
14. Navarro, G., Mäkinen, V.: Compressed full text indexes. ACM Comput. Surv. **39**(1) (2007)
15. Raman, R., Raman, V., Rao, S.S.: Succinct indexable dictionaries with applications to encoding k-ary trees and multisets. In: Proc. 13th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 233–242. San Francisco, USA (2002)

Treewidth of Graphs

1987; Arnborg, Corneil, Proskurowski

Hans L. Bodlaender
Institute of Information and Computing Sciences
Algorithms and Complexity Group, Center for
Algorithmic Systems, University of Utrecht,
Utrecht, The Netherlands

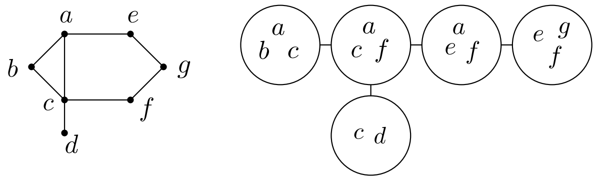
Keywords and Synonyms

Partial k-tree; Dimension; k-decomposable graphs

Problem Definition

The treewidth of graphs is defined in terms of tree decompositions. A *tree decomposition* of a graph $G = (V, E)$ is a pair $(\{X_i | i \in I\}, T = (I, F))$ with $\{X_i | i \in I\}$ a collection of subsets of V, called *bags*, and T a tree, such that

- $\bigcup_{i \in I} X_i = V$.
- For all $\{v, w\} \in E$, there is an $i \in I$ with $v, w \in X_i$.
- For all $v \in V$, the set $\{i \in I | v \in X_i\}$ induces a connected subtree of T.



Treewidth of Graphs, Figure 1
A graph and a tree decomposition of width 2

The *width* of a tree decomposition is $\max_{i \in I} |X_i| - 1$, and the treewidth of a graph G is the minimum width of a tree decomposition of G.

An alternative definition is in terms of chordal graphs. A graph $G = (V, E)$ is *chordal*, if and only if each cycle of length at least 4 has a chord, i. e., an edge between two vertices that are not successive on the cycle. A graph G has treewidth at most k, if and only if G is a subgraph of a chordal graph H that has maximum clique size at most k.

A third alternative definition is in terms of orderings of the vertices. Let π be a permutation (called *elimination scheme* in this context) of the vertices of $G = (V, E)$. Repeat the following step for $i = 1, \ldots, |V|$: take vertex $\pi(i)$, turn the set of its neighbors into a clique, and then remove v. The *width* of π is the maximum over all vertices of its degree when it was eliminated. The treewidth of G equals the minimum width over all elimination schemes.

In the treewidth problem, the given input is an undirected graph $G = (V, E)$, assumed to be given in its adjacency list representation, and a positive integer $k < |V|$. The problem is to decide if G has treewidth at most k, and if so, to give a tree decomposition of G of width at most k.

Key Results

Theorem 1 (Arnborg et al. [1]) *The problem, given a graph G and an integer k, is to decide if the treewidth of G of at most k is nondeterministic polynomial-time (NP) complete.*

For many applications of treewidth and tree decompositions, the case where k is assumed to be a fixed constant is very relevant. Arnborg et al. [1] gave in 1987 an algorithm that solves this problem in $O(n^{k+2})$ time. A number of faster algorithms for the problem with k fixed have been found; see, e. g., [6] for an overview.

Theorem 2 (Bodlaender [4]) *For each fixed k, there is an algorithm, that given a graph $G = (V, E)$ and an integer k, decides if the treewidth of G is at most k, and if so, that finds a tree decomposition of width at most k in O(n) time.*

This result of Theorem 2 is of theoretical importance only: in a practical setting, the algorithm appears to be much too slow owing to the large constant factor, hidden in the O-notation. For treewidth 1, the problem is equivalent to recognizing trees. Efficient algorithms based on a small set of reduction rules exist for treewidth 2 and 3 [2].

Two often-used heuristics for treewidth are the *minimum fill-in* and *minimum degree* heuristic. In the *minimum degree* heuristic, a vertex v of minimum degree is chosen. The graph G', obtained by making the neighborhood of v a clique and then removing v and its incident edges, is built. Recursively, a chordal supergraph H' of G' is made with the heuristic. Then, a chordal supergraph H of G is obtained, by adding v and its incident edges from G to H'. The *minimum fill-in* heuristic works similarly, but now a vertex is selected such that the number of edges that is added to make the neighborhood of v a clique is as small as possible.

Theorem 3 (Fomin et al. [9]) *There is an algorithm that, given a graph $G = (V, E)$, determines the treewidth of G and finds a tree decomposition of G of minimum width that uses $O(1.8899^n)$ time.*

Bouchitté and Todinca [8] showed that the treewidth can be computed in polynomial time for graphs that have a polynomial number of minimal separators. This implies polynomial-time algorithms for several classes of graphs, e. g., permutation graphs, weakly triangulated graphs.

Applications

One of the main applications of treewidth and tree decomposition is that many problems that are intractable (e. g., NP-hard) on arbitrary graphs become polynomial time or linear time solvable when restricted to graphs of bounded treewidth. The problems where this technique can be applied include many of the classic graph and network problems, like Hamiltonian circuit, Steiner tree, vertex cover, independent set, and graph coloring, but it can also be applied to many other problems. It is also used in the algorithm by Lauritzen and Spiegelhalter [11] to solve the inference problem on probabilistic ("Bayesian", or "belief") networks. Such algorithms typically have the following form. First, a tree decomposition of bounded width is found, and then a dynamic programming algorithm is run that uses this tree decomposition. Often, the running time of this dynamic programming algorithm is exponential in the width of the tree decomposition that is used, and thus one wants to have a tree decomposition whose width is as small as possible.

There are also general characterizations of classes of problems that are solvable in linear time on graphs of bounded treewidth. Most notable is the class of problems that can be formulated in *monadic second order logic* and extensions of these.

Treewidth has been used in the context of several applications or theoretical studies, including graph minor theory, data bases, constraint satisfaction, frequency assignment, compiler optimization, and electrical networks.

Open Problems

There are polynomial-time approximation algorithms for treewidth that guarantee a width of $O(k\sqrt{\log k})$ for graphs of treewidth k, but it is an open question whether there is a polynomial-time approximation algorithm for treewidth with a constant quality ratio. Another long-standing open problem is whether there is a polynomial-time algorithm to compute the treewidth of planar graphs.

Also open is to find an algorithm for the case where the bound on the treewidth k is fixed and whose running time as a function on n is polynomial, and as a function on k improves significantly on the algorithm of Theorem 2.

The base of the exponent of the running time of the algorithm of Theorem 3 can possibly be improved.

Experimental Results

Many algorithms (upper-bound heuristics, lower-bound heuristics, exact algorithms, and preprocessing methods) for treewidth have been proposed and experimentally evaluated. An overview of many of such results is given in [7]. A variant of the algorithm by Arnborg et al. [1] was implemented by Shoikhet and Geiger [15]. Röhrig [14] has experimentally evaluated the linear-time algorithm of Bodlaender [4], and established that it is not practical, even for small values of k. The *minimum degree* and *minimum fill-in* heuristics are frequently used [10].

Data Sets

A collection of test graphs and results for many of the algorithms on these graphs can be found in the TreewidthLIB collection [16].

Cross References

► Branchwidth of Graphs

Recommended Reading

1. Arnborg, S., Corneil, D.G., Proskurowski, A.: Complexity of finding embeddings in a k-tree. SIAM J. Algebr. Discret. Methods **8**, 277–284 (1987)

2. Arnborg, S., Proskurowski, A.: Characterization and recognition of partial 3-trees. SIAM J. Algebr. Discret. Methods **7**, 305–314 (1986)
3. Bodlaender, H.L.: A tourist guide through treewidth. Acta Cybernetica **11**, 1–23 (1993)
4. Bodlaender, H.L.: A linear time algorithm for finding tree-decompositions of small treewidth. SIAM J. Comput. **25**, 1305–1317 (1996)
5. Bodlaender, H.L.: A partial *k*-arboretum of graphs with bounded treewidth. Theor. Comp. Sci. **209**, 1–45 (1998)
6. Bodlaender, H.L.: Discovering treewidth. In: P. Vojtáš, M. Bieliková, B. Charron-Bost (eds.) Proceedings 31st Conference on Current Trends in Theory and Practive of Computer Science, SOFSEM 2005. Lecture Notes in Computer Science, vol. 3381, pp. 1–16. Springer, Berlin (2005)
7. Bodlaender, H.L.: Treewidth: Characterizations, applications, and computations. In: Fomin, F.V. (ed.) Proceedings 32nd International Workshop on Graph-Theoretic Concepts in Computer Science WG'06. Lecture Notes in Computer Science, vol. 4271, pp. 1–14. Springer, Berlin (2006)
8. Bouchitté, V., Todinca, I.: Listing all potential maximal cliques of a graph. Theor. Comput. Sci. **276**, 17–32 (2002)
9. Fomin, F.V., Kratsch, D., Todinca, I., Villanger, I.: Exact (exponential) algorithms for treewidth and minimum fill-in (2006). To appear in SIAM Journal of Computing, Preliminary version appeared in ICALP 2004
10. Koster, A.M.C.A., Bodlaender, H.L., van Hoesel, S.P.M.: Treewidth: Computational experiments. In: Broersma, H., Faigle, U., Hurink, J., Pickl, S. (eds.) Electronic Notes in Discrete Mathematics, vol. 8, pp. 54–57. Elsevier, Amsterdam (2001)
11. Lauritzen, S.J., Spiegelhalter, D.J.: Local computations with probabilities on graphical structures and their application to expert systems. J. Royal Stat. Soc. Ser. B (Methodological) **50**, 157–224 (1988)
12. Reed, B.A.: Tree width and tangles, a new measure of connectivity and some applications, LMS Lecture Note Series, vol. 241, pp. 87–162. Cambridge University Press, Cambridge (1997)
13. Reed, B.A.: Algorithmic aspects of tree width, pp. 85–107. CMS Books Math. Ouvrages Math. SMC, 11. Springer, New York (2003)
14. Röhrig, H.: Tree decomposition: A feasibility study. Master's thesis, Max-Planck-Institut für Informatik, Saarbrücken, Germany (1998)
15. Shoikhet, K., Geiger, D.: A practical algorithm for finding optimal triangulations. In: Proc. National Conference on Artificial Intelligence (AAAI '97), pp. 185–190. Morgan Kaufmann, San Fransisco (1997)
16. Bodlaender, H.L.: Treewidthlib. http://www.cs.uu.nl/people/hansb/treewidthlib (2004)

Triangle Finding

▶ Quantum Algorithm for Finding Triangles

Trip Planner

▶ Shortest Paths Approaches for Timetable Information

Truthful

▶ Nash Equilibria and Dominant Strategies in Routing

Truthful Auctions

▶ Truthful Mechanisms for One-Parameter Agents

Truthful Mechanisms for One-Parameter Agents

2001; Archer, Tardos

MOSHE BABAIOFF
Microsoft Research, Silicon Valley,
Mountain View, CA, USA

Keywords and Synonyms

Incentive compatible mechanisms; Dominant strategy mechanisms; Single-parameter agents; Truthful auctions

Problem Definition

This problem is concerned with designing truthful (dominant strategy) mechanisms for problems where each agent's private information is expressed by a single positive real number. The goal of the mechanisms is to allocate loads placed on the agents, and an agent's private information is the cost she incurs per unit load. Archer and Tardos [4] give an exact characterization for the algorithms that can be used to design truthful mechanisms for such load-balancing problems using appropriate side payments. The characterization shows that the allocated load must be monotonic in the cost (decreasing when the cost on an agent increases, fixing the costs of the others). Thus, truthful mechanisms are characterized by a condition on the allocation rule and on payments that ensures voluntary participation can be calculated using the given characterization.

The characterization is used to design polynomial-time truthful mechanisms for several problems in combinatorial optimization to which the celebrated Vickrey-Clarke-Groves (VCG) mechanism does not apply. For scheduling related parallel machines to minimize makespan ($Q\|C_{\max}$), Archer and Tardos [4] presented a 3-approximation mechanism based on randomized rounding of the optimal fractional solution. This mechanism is truthful only in expectation (a weaker notion of truthfulness in which truthful bidding maximizes the agent's *expected* utility). Archer [3] improved it to a randomized 2-approximation truthful mechanism. Andelman et al. [2] provided a deterministic truthful mechanism that is 5-approximation. Kovács improved it to 3-approximation in [9], and to 2.8-approximation in [10] (Kovács also gave other results for two special cases). Andelman et al. [2] also presented a deterministic fully polynomial time approximation scheme (FPTAS) for scheduling on a fixed number of machines, as well as a suitable payment scheme that yields a deterministic truthful mechanism. Archer and Tardos [4] presented results for goals other than minimizing the makespan. They presented a truthful mechanism for $Q\|\sum C_j$ (scheduling related machines to minimize the sum of completion times), and showed that for $Q\|\sum w_jC_j$ (minimizing the weighted sum of completion times) $2/\sqrt{3}$ is the best approximation ratio achievable by a truthful mechanism.

This family of problems belongs to the field of algorithmic mechanism design, initiated in the seminal paper of Nisan and Ronen [12]. Nisan and Ronen considered makespan minimization for scheduling on *unrelated* machines and proved upper and lower bounds (note that for unrelated machines agents have more than one parameter). Mu'alem and Schapira [11] presented improved lower bounds. The problem of scheduling on related machines to minimize the makespan has been considered in other papers. Auletta et al. [5] and Ambrosio and Auletta [1] presented truthful mechanisms for several nondeterministic polynomial-time hard restrictions of this problem. Nisan and Ronen [12] also introduced a model in which the mechanism is allowed to observe the machines' actual processing time and compute the payments afterwards (in such a model the machines essentially cannot claim to be faster than they are). Auletta et al. [6] presented additional results for this model. In particular, they showed that it is possible to overcome the lower bound of $2/\sqrt{3}$ for $Q\|\sum w_jC_j$ (minimizing the weighted sum of completion times) and provided a polynomial-time $(1+\epsilon)$-approximation truthful mechanism (with verification) when the number of machines (m) is constant.

The Mechanism Design Framework

Let I be the set of agents. Each agent $i \in I$ has some private *value* (type) consisting of a single parameter $t_i \in \Re$ that describes the agent, and which only i knows. Everything else is public knowledge. Each agent will report a *bid* b_i to the mechanism. Let t denote the vector of true values, and b the vector of bids.

There is some set of *outcomes* O, and given the bids b the mechanism's output algorithm computes an outcome $o(b) \in O$. For any types t, the mechanism aims to choose an outcome $o \in O$ that minimizes some function $g(o, t)$. Yet, given the bids b the mechanism can only choose the outcome as a function of the bids ($o = o(b)$) and has no knowledge of the true types t. To overcome the problem that the mechanism knows only the bids b, the mechanism is designed to be truthful (using payments), that is, in such a mechanism it is a dominant strategy for the agents to reveal their true types ($b = t$). For such mechanisms minimizing $g(o, t)$ is done by assuming that the bids are the true types (and this is justified by the fact that truth-telling is a dominant strategy).

In the framework discussed here we assume that outcome $o(b)$ will assign some amount of *load* or work $w_i(o(b))$ to each agent i, and given $o(b)$ and t_i, agent i incurs some monetary *cost*, $cost_i(t_i, o(b)) = t_i w_i(o(b))$. Thus, agent i's private data t_i measure her cost per unit work.

Each agent i attempts to maximize her *utility* (profit), $u_i(t_i, b) = P_i(b) - cost_i(t_i, o(b))$, where $P_i(b)$ is the *payment* to agent i.

Let b_{-i} denote the vector of bids, not including agent i, and let $b = (b_{-i}, b_i)$. Truth-telling is a *dominant strategy* for agent i if bidding t_i always maximizes her utility, regardless of what the other agents bid. That is, $u_i(t_i, (b_{-i}, t_i)) \geq u_i(t_i, (b_{-i}, b_i))$ for all b_{-i} and b_i.

A mechanism M consists of the pair $M = (o(\cdot), P(\cdot))$, where $o(\cdot)$ is the *output function* and $P(\cdot)$ is the *payment scheme*, i. e., the vector of payment functions $P_i(\cdot)$. An output function *admits a truthful payment scheme* if there exist payments $P(\cdot)$ such that for the mechanism $M = (o(\cdot), P(\cdot))$, truth-telling is a dominant strategy for each agent. A mechanism that admits a truthful payment scheme is *truthful*.

Mechanism M satisfies the *voluntary participation* condition if agents who bid truthfully never incur a net loss, i. e., $u_i(t_i, (b_{-i}, t_i)) \geq 0$ for all agents i, true values t_i, and other agents' bids b_{-i}.

Definition 1 With the other agents' bids b_{-i} fixed, the *work curve* for agent i is $w_i(b_{-i}, b_i)$, considered as a single-variable function of b_i. The output function o is *decreasing* if each of the associated work curves is decreasing (i. e., $w_i(b_{-i}, b_i)$ is a decreasing function of b_i, for all i and b_{-i}).

Scheduling on Related Machines

There are n jobs and m machines. The jobs represent amounts of work $p_1 \geq p_2 \geq \cdots \geq p_n$, and let p denote the set of jobs. Machine i runs at some speed s_i, so it must spend p_j/s_i units of time processing each job j assigned to it. The input to an algorithm is b, the (reported) speed of the machines, and the output is $o(b)$, an assignment of jobs to machines. The load on machine i for outcome $o(b)$ is $w_i(b) = \sum p_j$, where the sum runs over jobs j assigned to i. Each machine incurs a cost proportional to the time it spends processing its jobs. The cost of machine i is $cost_i(t_i, o(b)) = t_i w_i(o(b))$, where $t_i = 1/s_i$ and $w_i(b)$ is the total load assigned to i when the speeds are b. Let C_j denote the completion time of job j. One can consider the following goals for scheduling related parallel machines:

- Minimizing the makespan ($Q\|C_{\max}$), the mechanism's goal is to minimize the completion time of the last job on the last machine, i. e., $g(o, t) = C_{\max} = \max_i t_i \cdot w_i(b)$.
- Minimize the sum of completion times ($Q\| \sum C_j$), i. e. $g(o, t) = Q\| \sum C_j = \sum_j C_j$.
- Minimize the weighted sum of completion times ($Q\| \sum w_j C_j$), i. e., $g(o, t) = Q\| \sum w_j C_j = \sum_j w_j C_j$, where w_j is the weight of job j.

An algorithm is a *c-approximation algorithm* with respect to g, if for every instance (p, t) it outputs an outcome of cost at most $c \cdot g(o(t), t)$. A *c-approximation mechanism* is one whose output algorithm is a c-approximation. Note that if the mechanism is truthful the approximation is with respect to the true speeds. A polynomial-time approximation scheme (PTAS) is a family of algorithms such that for every $\epsilon > 0$ there exists a $(1 + \epsilon)$-approximation algorithm. If the running time is also polynomial in $1/\epsilon$, the family of algorithms is a FPTAS.

Key Results

The following two theorems hold for the mechanism design framework as defined in Sect. Problem Definition.

Theorem 1 ([4]) *The output function o(b) admits a truthful payment scheme if and only if it is decreasing. In this case, the mechanism is truthful if and only if the payments $P_i(b_{-i}, b_i)$ are of the form*

$$h_i(b_{-i}) + b_i w_i(b_{-i}, b_i) - \int_0^{b_i} w_i(b_{-i}, u)\,\mathrm{d}u \,,$$

where the h_i are arbitrary functions.

Theorem 2 ([4]) *A decreasing output function admits a truthful payment scheme satisfying voluntary participation if and only if $\int_0^\infty w_i(b_{-i}, u)du < \infty$ for all i, b_{-i}. In this case, the payments can be defined by*

$$P_i(b_{-i}, b_i) = b_i w_i(b_{-i}, b_i) + \int_{b_i}^\infty w_i(b_{-i}, u)\,\mathrm{d}u \,.$$

Theorem 3 ([4]) *There is a truthful mechanism (not polynomial time) that outputs an optimal solution for $Q\|C_{max}$ and satisfies voluntary participation.*

Theorem 4 *For the problem of minimizing the makespan ($Q\|C_{max}$):*

- *There is a polynomial-time randomized algorithm that deterministically yields a 2-approximation, and admits a truthful payment scheme that creates a mechanism that is truthful in expectation and satisfies voluntary participation [3] .*
- *There is a polynomial-time deterministic 2.8-approximation algorithm that admits a truthful payment scheme that creates a mechanism that satisfies voluntary participation [10].*
- *There is a deterministic FPTAS for scheduling on a fixed number of machines that admits a truthful payment scheme that creates a mechanism that satisfies voluntary participation [2].*

Theorem 5 ([4]) *There is a truthful polynomial-time mechanism that outputs an optimal solution for $Q\| \sum C_j$ and satisfies voluntary participation.*

Theorem 6 ([4]) *No truthful mechanism for $Q\| \sum w_j C_j$ can achieve an approximation ratio better than $2/\sqrt{3}$, even on instances with just two jobs and two machines.*

Applications

Archer and Tardos [4] applied the characterization of truthful mechanisms to problems other than scheduling. They presented results for the uncapacitated facility location problem as well as the maximum-flow problem.

Kis and Kapolnai [8] considered the problem of scheduling of groups of identical jobs on related machines with sequence-independent setup times ($Q|u_j, p_{jk} = p_j\|C_{\max}$). They provided a truthful, polynomial-time, randomized mechanism for the batch-scheduling problem with a deterministic approximation guarantee of 4 to the minimal makespan, based on the characterization of truthful mechanisms presented above.

Open Problems

Considering scheduling on related machines to minimize the makespan, Hochbaum and Shmoys [7] presented a PTAS for this problem, but it is not monotonic. Is there a truthful PTAS for this problem when the number of machines is not fixed? It is still an open problem whether such a mechanism exists or not. Finding such a mechanism would be an interesting result. Proving a lower bound that shows that such a mechanism does not exist would be even more interesting as it will show that there is a "cost of truthfulness" for this computational problem. A gap between the best approximation algorithm and the best monotonic algorithm (which creates a truthful mechanism), if it exists for this problem, would be a major step in improving our understanding of the combined effect of computational and incentive constraints.

Cross References

► Algorithmic Mechanism Design
► Competitive Auction
► Generalized Vickrey Auction
► Incentive Compatible Selection

Recommended Reading

1. Ambrosio, P., Auletta, V.: Deterministic monotone algorithms for scheduling on related machines. In: 2nd Ws. on Approx. and Online Alg. (WAOA), 2004, pp. 267–280
2. Andelman, N., Azar, Y., Sorani, M.: Truthful approximation mechanisms for scheduling selfish related machines. In: 22nd Ann. Symp. on Theor. Aspects of Comp. Sci. (STACS), 2005, pp. 69–82
3. Archer, A.: Mechanisms for Discrete Optimization with Rational Agents. Ph. D. thesis, Cornell University (2004)
4. Archer, A., Tardos, É.: Truthful mechanisms for one-parameter agents. In: 42nd Annual Symposium on Foundations of Computer Science (FOCS), 2001, pp. 482–491
5. Auletta, V., De Prisco, R., Penna, P., Persiano, G.: Deterministic truthful approximation mechanisms for scheduling related machines. In: 21st Ann. Symp. on Theor. Aspects of Comp. Sci. (STACS), 2004, pp. 608–619
6. Auletta, V., De Prisco, R., Penna, P., Persiano, G., Ventre, C.: New constructions of mechanisms with verification. In: 33rd International Colloquium on Automata, Languages and Programming (ICALP) (1), 2006, pp. 596–607
7. Hochbaum, D., Shmoys, D.: A polynomial approximation scheme for scheduling on uniform processors: Using the dual approximation approach. SIAM J. Comput. **17**(3), 539–551 (1988)
8. Kis, T., Kapolnai, R.: Approximations and auctions for scheduling batches on related machines. Operat. Res. Let. **35**(1), 61–68 (2006)
9. Kovács, A.: Fast monotone 3-approximation algorithm for scheduling related machines. In: 13th Annual European Symposium (ESA), 2005, pp. 616–627
10. Kovács, A.: Fast Algorithms for Two Scheduling Problems. Ph. D. thesis, Universität des Saarlandes (2007)
11. Mu'alem, A., Schapira, M.: Setting lower bounds on truthfulness. In: SODA, 2007
12. Nisan, N., Ronen, A.: Algorithmic mechanism design. Game. Econ. Behav. **35**, 166–196 (2001)

Truthful Multicast

2004; Wang, Li, Wang

WEIZHAO WANG[1], XIANG-YANG LI[2], YU WANG[3]
[1] Google Inc., Irvine, CA, USA
[2] Department of Computer Science, Illinois Institute of Tech., Chicago, IL, USA
[3] Department of Computer Science, University of North Carolina at Charlotte, Charlotte, NC, USA

Keywords and Synonyms

Truthful multicast routing; Strategyproof multicast mechanism

Problem Definition

Several mechanisms [1,3,5,9], which essentially all belong to the VCG mechanism family, have been proposed in the literature to prevent the selfish behavior of unicast routing in a wireless network. In these mechanisms, the least cost path, which maximizes the social efficiency, is used for routing. Wang, Li, and Wang [8] studied the *truthful* multicast routing protocol for a selfish wireless network, in which selfish wireless terminals will follow their own interests. The multicast routing protocol is composed of two components: (1) the tree structure that connects the sources and receivers, and (2) the payment to the relay nodes in this tree. Multicast poses a unique challenge in designing strategyproof mechanisms due to the reason that (1) a VCG mechanism uses an output that maximizes the *social efficiency*; (2) it is NP-hard to find the tree structure with the minimum cost, which in turn maximizes the social efficiency. A range of multicast structures, such as the least cost path tree (LCPT), the pruning minimum spanning tree (PMST), virtual minimum spanning tree (VMST), and Steiner tree, were proposed to replace the optimal multicast tree. In [8], Wang et al. showed how payment schemes can be designed for existing multicast tree structures so that rational selfish wireless terminals will follow the protocols for their own interests.

Consider a communication network $G = (V, E, c)$, where $V = \{v_1, \cdots, v_n\}$ is the set of communication terminals, $E = \{e_1, e_2, \cdots, e_m\}$ is the set of links, and c is the cost vector of all agents. Here agents are terminals in a node weighted network and are links in a link weighted network. Given a set of sources and receivers $Q = \{q_0, q_1, q_2, \cdots, q_{r-1}\} \subset V$, the multicast problem is to find a tree $T \subset G$ spanning all terminals Q. For simplicity, assume that $s = q_0$ is the sender of a multicast session if it exists. All terminals or links are required to declare a cost of relaying the message. Let d be the declared costs of all nodes, i. e., agent i declared a cost d_i. On the basis of the declared cost profile d, a multicast tree needs to be constructed and the payment $p_k(d)$ for each agent k needs to be decided. The utility of an agent is its payment received, minus its cost if it is selected in the multicast tree. Instead of reinventing the wheels, Wang et al. still used the previously proposed structures for multicast as the output of their mechanism. Given a multicast tree, they studied the design of strategyproof payment schemes based on this tree.

Notations

Given a network H, $\omega(H)$ denotes the total cost of all agents in this network. If the cost of any agent i (link e_i or node v_i) is changed to c'_i, the new network is denoted as $G' = (V, E, c|^i c'_i)$, or simply $c|^i c'_i$. If one agent i is removed from the network, it is denoted as $c|^i\infty$. For the simplicity of notation, the cost vector c is used to denote the network $G = (V, E, c)$ if no confusion is caused. For a given source s and a given destination q_i, $\mathsf{LCP}(s, q_i, c)$ represents the shortest path between s and q_i when the cost of the network is represented by vector c. $|\mathsf{LCP}(s, q_i, d)|$ denotes the total cost of the least cost path $\mathsf{LCP}(s, q_i, d)$. The notation of several multicast trees is summarized as follows.

1. Link Weighted Multicast Tree
   - **LCPT**: The union of all least cost paths from the source to receivers is called the *least cost path tree*, denoted by $LCPT(d)$.
   - **PMST**: First construct the minimum spanning tree $MST(G)$ on the graph G. Take the tree $MST(G)$ rooted at sender s, prune all subtrees that do not contain a receiver. The final structure is called the Pruning Minimum Spanning Tree (PMST).
   - **LST**: The Link Weighted Steiner Tree (LST) can be constructed by the algorithm proposed by Takahashi and Matsuyama [6].
2. Node Weighted Multicast Tree
   - **VMST**: First construct a virtual graph using all receivers plus the sources as the vertices and the cost of LCP as the link weight. Then compute the minimum spanning tree on the virtual graph, which is called virtual minimum spanning tree (VMST). Finally, choose all terminals on the VMST as the relay terminals.
   - **NST**: The node weighted Steiner tree (NST) can be constructed by the algorithm proposed by [4].

Key Results

If the LCPT tree is used as the multicast tree, Wang et al. proved the following theorem.

Theorem 1 *The VCG mechanism combined with LCPT is not truthful.*

Because of the failure of the VCG mechanism, they designed their non-VGC mechanism for the LCPT-based multicast routing as follows.

```
1: For each receiver $q_i \neq s$, computes the least cost path from the source $s$ to $q_i$, and compute a payment $p^i_k(d)$ to every link $e_k$ on the LCP$(s, q_i, d)$ using the scheme for unicast
       $p^i_k(d) = d_k + |\mathsf{LCP}(s, q_i, d|^k\infty)| - |\mathsf{LCP}(s, q_i, d)|$.
2: The final payment to link $e_k \in LCPT$ is then
       $p_k(d) = \max_{q_i \in Q} p^i_k(d)$.   (1)
   The payment to each link not on LCPT is simply 0.
```

Truthful Multicast, Algorithm 1
Non-VCG mechanism for LCPT

Theorem 2 *Payment (defined in Eq. (1)) based on LCPT is truthful and it is minimum among all truthful payments based on LCPT.*

More generally, Wang et al. [8] proved the following theorem.

Theorem 3 *The VCG mechanism combined with either one of the LCPT, PMST, LST, VMST, NST is not truthful.*

```
1: Apply VCG mechanism on the MST. The payment for edge $e_k \in PMST(d)$ is
       $p_k(d) = \omega(MST(d|^k\infty)) - \omega(MST(d)) + d_k$.   (2)
2: For every edge $e_k \notin PMST(d)$, its payment is 0.
```

Truthful Multicast, Algorithm 1
Non-VCG mechanism for PMST

Because of this negative result, they designed their non-VCG mechanisms for all multicast structures they studied: LCPT, PMST, LST, VMST, NST. For example, Algorithm 2 is the algorithm for PMST. For other algorithms, please refer to [8].

Regarding all their non-VGC mechanisms, they proved the following theorem.

Theorem 4 *The non-VCG mechanisms designed for the multicast structures LCPT, PMST, LST, VMST, NST are not only truthful, but also achieve the minimum payment among all truthful mechanisms.*

Applications

In wireless ad hoc networks, it is commonly assumed that, each terminal contributes its local resources to forward the data for other terminals to serve the common good, and benefits from resources contributed by other terminals to route its packets in return. On the basis of such a fundamental design philosophy, wireless ad hoc networks provide appealing features such as enhanced system robustness, high service availability and scalability. However, the critical observation that individual users who own these wireless devices are generally selfish and non-cooperative may severely undermine the expected performances of the wireless networks. Therefore, providing incentives to wireless terminals is a must to encourage contribution and thus maintains the robustness and availability of wireless networking systems. On the other hand, to support a communication among a group of users, multicast is more efficient than unicast or broadcast, as it can transmit packets to destinations using fewer network resources, thus increasing the social efficiency. Thus, most results of the work of Wang et al. can apply to multicast routing in wireless networks in which nodes are selfish. It not only guarantees that multicast routing behaves normally but also achieves good social efficiency for both the receivers and relay terminals.

Open Problems

There are several unsolved challenges left as future work in [8]. Some of these challenges are listed below.

- How to design algorithms that can compute these payments in asymptotically optimum time complexities is presently unknown.
- Wang et al. [8] only studied the tree-based structures for multicast. Practically, mesh-based structures may be more needed for wireless networks to improve the fault tolerance of the multicast. It is unknown whether a strategyproof multicast mechanism can be designed for some mesh-based structures used for multicast.
- All of the tree construction and payment calculations in [8] are performed in a centralized way, it would be interesting to design some distributed algorithms for them.
- In the work by Wang et al. [8] it was assumed that the receivers will always relay the data packets for other receivers for free, the source node of the multicast will pay the relay nodes to compensate their cost, and the source node will not charge the receivers for getting the data. As a possible future work, the budget balance of the source node needs to be considered if the receivers have to pay the source node for getting the data.
- Fairness of payment sharing needs to be considered in a case where the receivers share the total payments to all relay nodes on the multicast structure. Notice that this is different from the cost-sharing studied in [2], in which they assumed a fixed multicast tree, and the link cost is publicly known; in that work they showed how to share the total link cost among receivers.
- Another important task is to study how to implement the protocols proposed in [8] in a distributed manner. Notice that, in [3,9], distributed methods have been developed for a truthful unicast using some cryptography primitives.

Cross References

► Algorithmic Mechanism Design

Recommended Reading

1. Anderegg, L., Eidenbenz, S.: Ad hoc-VCG: a truthful and cost-efficient routing protocol for mobile ad hoc networks with selfish agents. In: Proceedings of the 9th annual international conference on Mobile computing and networking. pp. 245–259 ACM Press, New York (2003)
2. Feigenbaum, J., Papadimitriou, C., Shenker, S.: Sharing the cost of multicast transmissions. J. Comput. Syst. Sci. **63**(1), 21–41 (2001)
3. Feigenbaum, J., Papadimitriou, C., Sami, R., Shenker, S.: A BGP-based mechanism for lowest-cost routing. In: Proceedings of the 2002 ACM Symposium on Principles of Distributed Computing, pp. 173–182. Monterey, 21–24 July 2002
4. Klein, P., Ravi, R.: A nearly best-possible approximation algorithm for node-weighted Steiner trees. J. Algorithms **19**(1), 104–115 (1995)
5. Nisan, N., Ronen, A.: Algorithmic mechanism design. In: Proc. 31st Annual Symposium on Theory of Computing (STOC99), Atlanta, 1–4 May 1999, pp. 129–140 (1999)
6. Takahashi, H., Matsuyama, A.: An approximate solution for the Steiner problem in graphs. Math. Jap. **24**(6), 573–577 (1980)
7. Wang, W., Li, X.-Y.: Low-Cost routing in selfish and rational wireless ad hoc networks. IEEE Trans. Mobile Comput. **5**(5), 596–607 (2006)
8. Wang, W., Li, X.-Y., Wang, Y.: Truthful multicast in selfish wireless networks. In: Proceedings of the 10th ACM Annual international Conference on Mobile Computing and Networking, Philadelphia, 26 September – 1 October 2004
9. Zhong, S., Li, L., Liu, Y., Yang, Y.R.: On designing incentive-compatible routing and forwarding protocols in wireless ad-hoc networks –an integrated approach using game theoretical and cryptographic techniques. In: Proceedings of the 11th ACM Annual international Conference on Mobile Computing and Networking, Cologne, 28 August – 2 September 2005

Truthful Multicast Routing

► Truthful Multicast

t-Spanners

► Geometric Spanners

TSP-Based Curve Reconstruction

2001; Althaus, Mehlhorn

Edgar Ramos
School of Mathematics, National University of Colombia, Medellín, Colombia

Problem Definition

An instance of the curve reconstruction problem is a finite set of *sample* points V in the plane, which are assumed to be taken from an unknown planar curve γ. The task is to construct a geometric graph G on V such that two points in V are connected by an edge in G if and only if the points are adjacent on γ. The curve γ may consist of one or more connected components, and each of them may be closed or open (with endpoints), and may be smooth everywhere (tangent defined at every point) or not.

Many heuristic approaches have been proposed to solve this problem. This work continues a line of reconstruction algorithms with *guaranteed* performance, i. e. algorithms which probably solve the reconstruction problem under certain assumptions of γ and V. Previous proposed solutions with guaranteed performances were mostly *local*: a subgraph of the complete geometric graph defined by the points is considered (in most cases the Delaunay edges), and then *filtered* using a local criteria into a subgraph that will constitute the reconstruction. Thus, most of these algorithms fail to enforce that the solution have the global property of being a path/tour or collection of paths/tours and so usually require a dense sampling to work properly and have difficulty handling nonsmooth curves. See [6,7,8] for surveys of these algorithms.

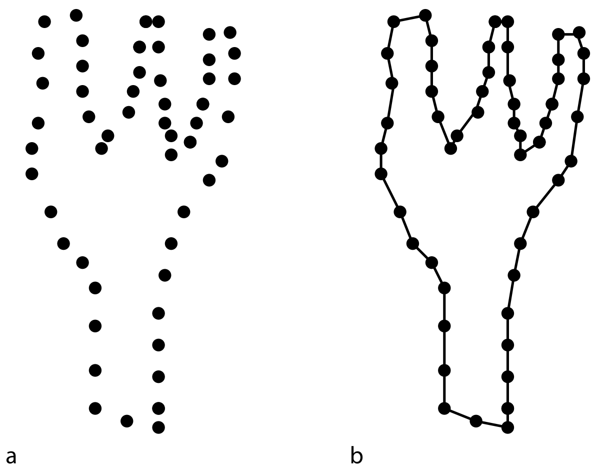
This work concentrates on a solution approach based on the *traveling salesman problem (TSP)*. Recall that a *traveling salesman path (tour)* for a set V of points is a path (cycle) passing through all points in V. An optimal traveling salesman path (tour) is a traveling salesman path (tour) of shortest length. The first question is under which conditions for γ and V a traveling salesman path (tour) is a correct reconstruction. Since the construction of an optimal traveling salesman path (tour) is an NP-hard problem, a second question is whether for the specific instances under consideration, an efficient algorithm is possible.

A previous work of Giesen [9] gave a first weak answer to the first question: For every benign semiregular closed curve γ, there exists an $\epsilon > 0$ with the following property: If V is a finite sample set from γ so that for every $x \in \gamma$ there is a $p \in V$ with $\|pv\| \le \epsilon$, then the optimal traveling salesman tour is a polygonal reconstruction of γ. For a curve $\gamma : [0,1] \to R^2$, its left and right tangents at $\gamma(t_0)$, are defined as the limits of the ratio $|\gamma(t_2) - \gamma(t_1)| \,/\, |t_2 - t_1|$ as (t_1, t_2) converges to (t_0, t_0) from the right $(t_0 < t_1 < t_2)$ and from the right $(t_1 < t_2 < t_0)$ respectively. A curve is semiregular if both tangents exist at every points and regular if the tangents exist and coincide at every point. The *turning* angle of γ at p is the angle between the left and right tangents at a points p. A semiregular curve is *benign* if the turning angle is less than π.

To investigate the TSP-based solution of the reconstruction problem, this work considers its integer linear programming *(ILP)* formulation and the corresponding linear programming *(LP)* relaxation. The motivation is that a successful method for solving the TSP is to use a branch-and-cut algorithm based on the LP-relaxation. See Chapter 7 in [5]. For a path with endpoints a and b, the formulation is based on variables $x_{u,v} \in \{0,1\}$ for each pair u, v in V (indicating whether the edge uv is in the path $(x_{uv} = 1)$ or not $(x_{uv} = 0)$ and consists of the following objective function and constraints ($x_{uu} = 0$ for all $u \in V$):

$$\begin{array}{lll}
\text{minimize} & \displaystyle\sum_{u,v\in V} \|uv\| \cdot x_{uv} & \\
\text{subject to} & \displaystyle\sum_{v\in V} x_{uv} = 2 & \text{for all } u \in V \setminus \{a,b\} \\
& \displaystyle\sum_{v\in V} x_{uv} = 1 & \text{for } u \in \{a,b\} \\
& \displaystyle\sum_{u,v\in V'} x_{uv} \le |V'| - 1 & \text{for } V' \subseteq V,\ V' \ne \emptyset \\
& x_{uv} \in \{0,1\} & \text{for all } u, v \in V.
\end{array}$$

Here $\|uv\|$ denotes the Euclidean distance between u and v and so the objective function is the total length of the selected edges. This is called the *subtour-ILP for the TSP with specified endpoints*. The equality constraints are called the *degree constraints*, the inequality ones are called *subtour elimination constraints* and the last ones are called the *integrality constraints*. If the degree and integrality constraints hold, the corresponding graph could include disconnected cycles (subtours), hence the need for the subtour elimination constraints. The relaxed LP is obtained by replacing



TSP-Based Curve Reconstruction, Figure 1
Sample data and its reconstruction

the integrality constraints by the constraints $0 \le x_{uv} \le 1$ and is called the *subtour-LP for the TSP with specified endpoints*. There is a polynomial time algorithm that given a candidate solution returns a violated constraint if it exists: the degree constraints are trivial to check and the subtour elimination constraints are checked using a min cut algorithm (if a, b are joined by an edge and all edge capacities are made equal to one, then a violated subtour constraint corresponds to a cut smaller than two). This means that the subtour-LP for the TSP with specified endpoints can potentially be solved in polynomial time in the bit size of the input description, using the ellipsoid method [10].

Key Results

The main results of this paper are that, given a sample set V with $a, b \in V$ from a benign semiregular open curve γ with endpoints a, b and satisfying certain *sampling condition* [it], then

- the optimal traveling salesman path on V with endpoints a, b is a polygonal reconstruction of γ from V,
- the subtour-LP for traveling salesman paths has an optimal integral solution which is unique.

This means that, under the sampling conditions, the subtour-LP solution provides a TSP solution and also suggests a reconstruction algorithm: solve the subtour-LP and, if the solution is integral, output it. If the input satisfies the sampling condition, then the solution will be integral and the result is indeed a polygonal reconstruction. Two algorithms are proposed to solve the subtour-LP. First, using the simplex method and the cutting plane framework: it starts with an LP consisting of only the degree constraints and in each iteration solves the current LP and checks whether that solution satisfies all the subtour elimination constraints (using a min cut algorithm) and, if not, adds a violated constraint to the current LP. This algorithm has a potentially exponential running time. Second, using a similar approach but with the ellipsoid method. This can be implemented so that the running time is polynomial in the bit size of the input points. This requires justification for using approximate point coordinates and distances.

The main tool in deriving these results is the connection between the subtour-LP and the so-called *Held–Karp bound*. The line of argument is as follows:

- Let $c(u, v) = \|uv\|$ and $\mu : V \to R$ be a *potential function*. The corresponding *modified distance function* c_μ is defined by $c_\mu(u, v) = c(u, v) - \mu(u) - \mu(v)$.
- For any traveling salesman path T with endpoints a, b,
$$c_\mu(T) = c(T) - 2\sum_{v \in V} \mu(v) + \mu(a) + \mu(b),$$
and so an optimal traveling salesman path with endpoints a, b for c_μ is also optimal for c.
- Let C_μ be the cost of a minimum spanning tree MST_μ under c_μ, then since a traveling salesman path is a spanning tree, the optimal traveling salesman T_0 satisfies $C_\mu \le c_\mu(T_0) = c(T_0) - 2\sum_{v \in V} \mu(v) + \mu(a) + \mu(b)$, and so
$$\max_\mu \left(C_\mu + 2\sum_{v \in V} \mu(v) - \mu(a) - \mu(b) \right) \le c(T_0)\,.$$
The term on the left is the so called Held–Karp bound.
- Now, if for a particular μ, MST_μ is a path with endpoints a, b, then MST_μ is in fact an optimal traveling salesman path with endpoints a, b, and the Held–Karp bound matches $c(T_0)$.
- The Held–Karp bound is equal to the optimal objective value of the subtour-LP. This follows by relaxation of the degree constraints in a Lagrangian fashion (see [5]) and gives an effective way to compute the Held-Karp bound: solve the subtour-LP.
- Finally, a potential function μ is constructed for γ so that, for an appropriately dense sample set V, MST_μ is unique and is a polygonal reconstruction with endpoints a, b. This then implies that solving the subtour-LP will produce a correct polygonal reconstruction.

Note that the potential function μ enters the picture only as an analysis tool. It is not needed by the algorithm. The authors extend this work to the case of open curves without specified endpoints and of closed curves using variations of the ILP formulation and a more restricted sampling condition. They also extend it to the case of a collection of closed curves. The latter requires preprocessing that partitions points into groups that are expected to form individual curves. Then each subgroup is processed with the subtour-LP approach and then the quality of the result assessed and then that partition may be updated.

Finite Precision

The above results are obtained assuming exact representation of point samples and the distances between them, so claiming a polynomial time algorithm is not immediate as the running time of the ellipsoid method is polynomial in the bit size of the input. The authors extend the results to the case in which points and the distances between them are known only approximately and from this they can conclude the polynomial running time.

Relation to Local Feature Size

The defined potential function μ is related to the so called *local feature size* function f used in the theory of smooth curve reconstruction, where $f(p)$ is defined as the distance from p to the medial axis of the curve γ. In this paper, $\mu(p)$ is defined as $d(p)/3$ where $d(p)$ is the size of the largest neighborhood of p so that γ in that neighborhood does not *deviate significantly* from a flat segment of curve. This paper shows $f(p) < 3d(p)$. In fact, $\mu(p)$ amounts to a generalization of the local feature size to nonsmooth curves (for a corner point p, $\mu(p)$ is proportional to the size of the largest neighborhood of p such that γ inside does not deviate significantly from a corner point with two nearly flat legs incident to it, and for points near the corner, μ is defined as an appropriate interpolation of the two definitions), and is in fact similar to definitions proposed elsewhere.

Applications

The curve reconstruction problem appears in applied areas such as cartography. For example, to determine level sets, features, object contours, etc. from samples. Admittedly, these applications usually may require the ability to handle very sparse sampling and noise. The 3D version of the problem is very important in areas such as industrial manufacturing, medical imaging, and computer animation. The 2D problem is often seen as a simpler (toy) problem to test algorithmic approaches.

Open Problems

A TSP-based solution when the curve γ is a collection of curves, not all closed, is not given in this paper. A solution similar to that for closed curves (partitioning and then application of subtour-LP for each) seems feasible for general collections, but some technicalities need to be solved. More interesting is the study of corresponding reconstruction approaches for surfaces in 3D.

Experimental Results

The companion paper [2] presents results of experiments comparing the TSP-based approach to several (local) Delaunay filtering algorithms. The TSP implementation uses the simplex method and the cutting plane framework (with a potentially exponential running time algorithm). The experiments show that the TSP-based approach has a better performance, allowing for much sparser samples than the others. This is to be expected given the global nature of the TSP-based solution. On the other hand, the speed of the TSP-based solution is reported to be competitive when compared to the speed of the others, despite its potentially bad worst-case behavior.

Data Sets

None reported. Experiments in [2] were performed with a simple reproducible curve based on a sinusoidal with varying number of periods and samples.

URL to Code

The code of the TSP-based solution as well as the other solutions considered in the companion paper [2] are available from: http://www.mpi-inf.mpg.de/~althaus/LEP:Curve-Reconstruction/curve.html

Cross References

- ► Engineering Geometric Algorithms
- ► Euclidean Traveling Salesperson Problem
- ► Minimum Weight Triangulation
- ► Planar Geometric Spanners
- ► Robust Geometric Computation

Recommended Reading

1. Althaus, E., Mehlhorn, K.: Traveling salesman-based curve reconstruction in polynomial time. SIAM J. Comput. **31**, 27–66 (2001)
2. Althaus, E., Mehlhorn, K., Näher, S., Schirra, S.: Experiments on curve reconstruction. In: ALENEX, 2000, pp. 103–114
3. Amenta, N., Bern, M.: Surface reconstruction by Voronoi filtering. Discret. Comput. Geom. **22**, 481–504 (1999)
4. Amenta, N., Bern, M., Eppstein, D.: The crust and the β-skeleton: Combinatorial curve reconstruction. Graph. Model. Image Process. **60**, 125–135 (1998)
5. Cook, W., Cunningham, W., Pulleyblank, W., Schrijver, A.: Combinatorial Optimization. Wiley, New York (1998)

6. Dey, T.K.: Curve and surface reconstruction. In: Goodman, J.E., O'Rourke, J. (eds.) Handbook of Discrete and Computational Geometry, 2nd edn. CRC, Boca Raton (2004)
7. Dey, T.K.: Curve and Surface Reconstruction: Algorithms with Mathematical Analysis. Cambridge University Press, New York (2006)
8. Edlesbrunner, H.: Shape reconstruction with the Delaunay complex. In: LATIN'98, Theoretical Informatics. Lecture Notes in Computer Science, vol. 1380, pp. 119–132. Springer, Berlin (1998)
9. Giesen, J.: Curve reconstruction, the TSP, and Menger's theorem on length. Discret. Comput. Geom. **24**, 577–603 (2000)
10. Schrijver, A.: Theory of Linear and Integer Programming. Wiley, New York (1986)

Two-Dimensional Compressed Matching

▶ Multidimensional Compressed Pattern Matching

Two-Dimensional Pattern Indexing

2005; Na, Giancarlo, Park

JOONG CHAE NA[1], PAOLO FERRAGINA[2],
RAFFAELE GIANCARLO[3], KUNSOO PARK[4]

[1] Department of Computer Science and Engineering, Sejong University, Seoul, Korea
[2] Department of Computer Science, University of Pisa, Pisa, Italy
[3] Department of Mathematics and Applications, University of Palermo, Palermo, Italy
[4] School of Computer Science and Engineering, Seoul National University, Seoul, Korea

Keywords and Synonyms

Two-Dimensional indexing for pattern matching; Two-Dimensional index data structures; Index data structures for matrices or images; Indexing for matrices or images

Problem Definition

This entry is concerned with designing and building indexes of a two-dimensional matrix, which is basically the generalization of indexes of a string, the *suffix tree* [12] and the *suffix array* [11], to a two-dimensional matrix. This problem was first introduced by Gonnet [7]. Informally, a two-dimensional analog of the suffix tree is a tree data structure storing all submatrices of an $n \times m$ matrix, $n \geq m$. The *submatrix tree* [2] is an incarnation of such indexes. Unfortunately, building such indexes requires $\Omega(nm^2)$ time [2]. Therefore, much of the attention paid has been restricted to square matrices and submatrices, the important special case in which much better results are available.

For square matrices, the *Lsuffix tree* and its array form, storing all *square submatrices* of an $n \times n$ matrix, have been proposed [3,9,10]. Moreover, the general framework for these index families is also introduced [4,5]. Motivated by LZ1-type image compression [14], the on-line case, i.e, the matrix is given one row or column at a time, has been also considered. These data structures can be built in time close to n^2. Building these data structures is a nontrivial extension of the algorithms for the standard suffix tree and suffix array. Generally, a tree data structure and its array form of this type for square matrices are referred to as the *two-dimensional suffix tree* and the *two-dimensional suffix array*, which are the main concerns of this entry.

Notations

Let A be an $n \times n$ matrix with entries defined over a finite alphabet Σ. $A[i..k, j..l]$ denotes the submatrix of A with corners (i,j), (k,j), (i,l), and (k,l). When $i = k$ or $j = l$, one of the repeated indexes is omitted. For $1 \leq i, j \leq n$, the *suffix* $A(i,j)$ of A is the largest square submatrix of A that starts at position (i,j) in A. That is, $A(i,j) = A[i..i+k, j..j+k]$ where $k = n - \max(i,j)$. Let $\$_i$ be a special symbol not in Σ such that $\$_i$ is lexicographically smaller than any other character in Σ. Assume that $\$_i$ is lexicographically smaller than $\$_j$ for $i < j$. For notational convenience, assume that the last entries of the ith row and column are $\$_i$. It makes all suffixes distinct. See Fig. 1a and b for an example.

Let $L\Sigma = \bigcup_{i=1}^{\infty} \Sigma^{2i-1}$. The strings of $L\Sigma$ are referred to as *Lcharacters*, and each of them is considered as an atomic item. $L\Sigma$ is called the *alphabet of Lcharacters*. Two Lcharacters are equal if and only if they are equal as strings over Σ. Moreover, given two Lcharacters La and Lb of equal length, La is *lexicographically smaller than or equal to* Lb if and only if the string corresponding to La is lexicographically smaller than or equal to that corresponding to Lb. A *chunk* is the concatenation of Lcharacters with the following restriction: an Lcharacter in Σ^{2i-1} can precede only one in $\Sigma^{2(i+1)-1}$ and succeed only one in $\Sigma^{2(i-1)-1}$. An *Lstring* is a chunk such that the first Lcharacter is in Σ.

For dealing with matrices as strings, a linear representation of square matrices is needed. Given $A[1..n, 1..n]$, divide A into n L-shaped characters. Let $a(i)$ be the concatenation of row $A[i, 1..i-1]$ and column $A[1..i, i]$. Then

$a(i)$ can be regarded as an Lcharacter. The linearized string of matrix A, called the Lstring of matrix A, is the concatenation of Lcharacters $a(1), \ldots, a(n)$. See Fig. 1c for an example. Slightly different linearizations have been used [9,10,13], but they are essentially the same in the aspect of two-dimensional functionality.

Two-Dimensional Suffix Trees

The suffix tree of matrix A is a compacted trie over the alphabet $L\Sigma$ that represents Lstrings corresponding to all suffixes of A. Formally, the *two-dimensional suffix tree* of matrix A is a rooted tree that satisfies the following conditions (see Fig. 1d for an example):

1. Each edge is labeled with a chunk.
2. There is no internal node of outdegree one.
3. Chunks assigned to sibling edges start with different Lcharacters, which are of the same length as strings in Σ^*.
4. The concatenation of the chunks labeling the edges on the path from the root to a leaf gives the Lstring of exactly one suffix of A, say $A(i, j)$. It is said that this leaf is associated with $A(i, j)$.
5. There is exactly one leaf associated with each suffix.

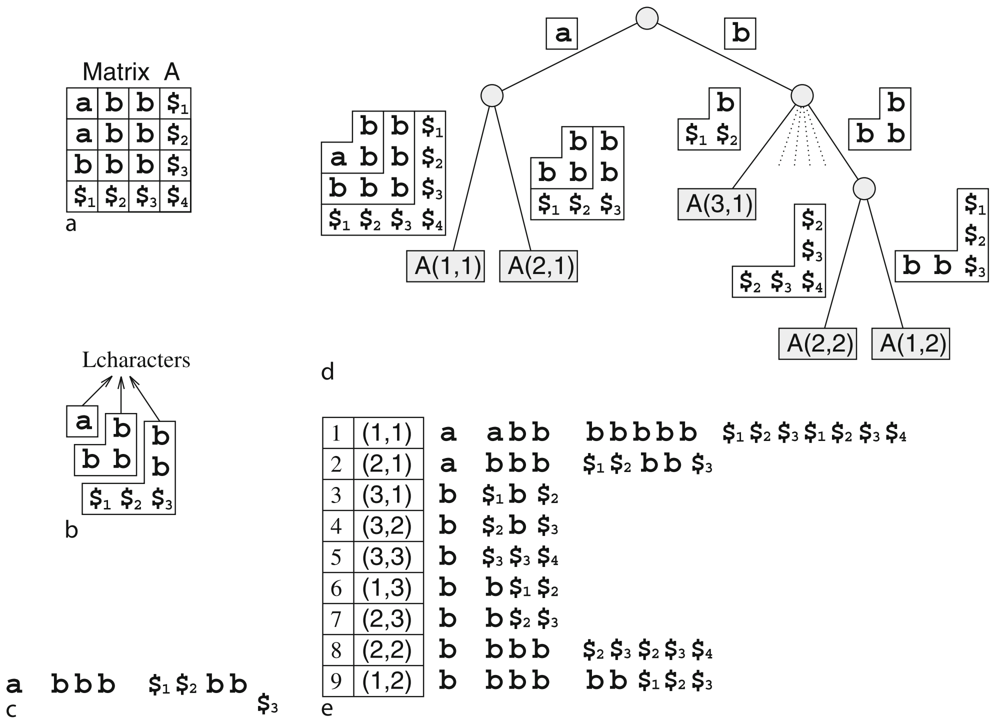
Conditions 4 and 5 mean that there is a one-to-one correspondence between the leaves of the tree and the suffixes of A (which are all distinct because $\$_i$ is unique).

Problem 1 (Construction of 2D suffix tree)
INPUT: *An $n \times n$ matrix A.*
OUTPUT: *A two-dimensional suffix tree storing all square submatrices of A.*

On-Line Suffix Trees

Assume that A is read *on-line* in row major order (column major order can be considered similarly). Let $A_t = A[1..t, 1..n]$ and $row_t = A[t, 1..n]$. At time $t-1$, nothing but A_{t-1} is known about A. At time t, row_t is read and so A_t is known. After time t, the on-line suffix tree of A is storing all suffixes of A_t. Note that Condition 4 may not be satisfied during the on-line construction of the suffix tree. A leaf may be associated with more than one suffix, because the suffixes of A_t are not all distinct.

Problem 2 (On-line construction of 2D suffix tree)
INPUT: *A sequence of rows of $n \times n$ matrix A, row_1, $row_2, \ldots, row_n$.*



1	(1,1)	a abb bbbbb $\$_1\$_2\$_3\$_1\$_2\$_3\$_4$
2	(2,1)	a bbb $\$_1\$_2$ bb $\$_3$
3	(3,1)	b $\$_1b\$_2$
4	(3,2)	b $\$_2b\$_3$
5	(3,3)	b $\$_3\$_3\$_4$
6	(1,3)	b b$\$_1\$_2$
7	(2,3)	b b$\$_2\$_3$
8	(2,2)	b bbb $\$_2\$_3\$_2\$_3\$_4$
9	(1,2)	b bbb bb$\$_1\$_2\$_3$

Two-Dimensional Pattern Indexing, Figure 1
a A matrix A, b the suffix $A(2,1)$ and Lcharacters composing $A(2,1)$, c the Lstring of $A(2,1)$, d the suffix tree of A, and e the suffix array of A (omitting the suffixes started with $\$_i$)

OUTPUT: *A two-dimensional suffix tree storing all square submatrices of A_t after reading row_t.*

Two-Dimensional Suffix Arrays

The *two-dimensional suffix array* of matrix A is basically a sorted list of all Lstrings corresponding to suffixes of A. Formally, the kth element of the array has the start position (i,j) if and only if the Lstring of $A(i,j)$ is the kth smallest one among the Lstrings of all suffixes of A. See Fig. 1e for an example. The two-dimensional suffix array is also coupled with additional information tables, called *Llcp* and *Rlcp*, to enhance its performance like the standard suffix array. The two-dimensional suffix array can be constructed from the two-dimensional suffix tree in linear time.

Problem 3 (Construction of 2D suffix array)
INPUT: *An $n \times n$ matrix A.*
OUTPUT: *The two-dimensional suffix array storing all square submatrices of A.*

Submatrix Trees

The *submatrix tree* is a tree data structure storing all submatrices. This entry just gives a result on submatrix trees. See [2] for details.

Problem 4 (Construction of a submatrix tree)
INPUT: *An $n \times m$ matrix B, $n \geq m$.*
OUTPUT: *The submatrix tree and its array form storing all submatrices of B.*

Key Results

Theorem 1 (Kim and Park 1999 [10], Cole and Hariharan 2000 [1]) *Given an $n \times n$ matrix A over an integer alphabet, one can construct the two-dimensional suffix tree in $O(n^2)$ time.*

Kim and Park's result is a deterministic algorithm, Cole and Hariharan's result is a randomized one. For an arbitrary alphabet, one needs first to sort it and then to apply the theorem above.

Theorem 2 (Na et al. 2005 [13]) *Given an $n \times n$ matrix A, one can construct on-line the two-dimensional suffix tree of A in $O(n^2 \log n)$ time.*

Theorem 3 (Kim et al. 2003 [9]) *Given an $n \times n$ matrix A, one can construct the two-dimensional suffix array of A in $O(n^2 \log n)$ time without constructing the two-dimensional suffix tree.*

Theorem 4 (Giancarlo 1993 [2]) *Given an $n \times m$ matrix B, one can construct the submatrix tree of B in $O(nm^2 \log(nm))$ time.*

Applications

Two-dimensional indexes can be used for many pattern-matching problems of two-dimensional applications such as low-level image processing, image compression, visual data bases, and so on [3,6]. Given an $n \times n$ text matrix and an $m \times m$ pattern matrix over an alphabet Σ, the *two-dimensional pattern retrieval problem*, which is a basic pattern matching problem, is to find all occurrences of the pattern in the text. The two-dimensional suffix tree and array of the text can be queried in $O(m^2 \log |\Sigma| + occ)$ time and $O(m^2 + \log n + occ)$ time, respectively, where occ is the number of occurrences of the pattern in the text. This problem can be easily extended to a set of texts. These queries have the same procedure and performance as those of indexes for strings. On-line construction of the two-dimensional suffix tree can be applied to LZ-1-type image compression [6].

Open Problems

The main open problems on two-dimensional indexes are to construct indexes in optimal time. The linear-time construction algorithm for two-dimensional suffix trees is already known [10]. The on-line construction algorithm due to [13] is optimal for unbounded alphabets, but not for integer or constant alphabets. Another open problem is to construct two-dimensional suffix arrays directly in linear time.

Experimental Results

An experiment that compares construction algorithms of two-dimensional suffix trees and suffix arrays was presented in [8]. Giancarlo's algorithm [2] and Kim et al.'s algorithm [8] were implemented for two-dimensional suffix trees and suffix arrays, respectively. Random matrices of sizes $200 \times 200 \sim 800 \times 800$ and alphabets of sizes 2, 4, 16 were used for input data. According to experimental results, the construction of two-dimensional suffix arrays is ten-times faster and five-times more space-efficient than that of two-dimensional suffix trees.

Cross References

► Multidimensional String Matching
► Suffix Array Construction
► Suffix Tree Construction in RAM

Recommended Reading

1. Cole, R. Hariharan, R.: Faster suffix tree construction with missing suffix links. In: Proceedings of the 30th Annual ACM Symposium on Theory of Computing, 2000, pp. 407–415
2. Giancarlo, R.: An index data structure for matrices, with applications to fast two-dimensional pattern matching. In: Proceed-

ings of Workshop on Algorithm and Data Structures, vol. 709, pp. 337–348. Springer Lect. Notes Comp. Sci. Montréal, Canada (1993)

3. Giancarlo, R.: A generalization of the suffix tree to square matrices, with application. SIAM J. Comput. **24**, 520–562 (1995)
4. Giancarlo, R., Grossi, R.: On the construction of classes of suffix trees for square matrices: Algorithms and applications. Inf. Comput. **130**, 151–182 (1996)
5. Giancarlo, R., Grossi, R.: Suffix tree data structures for matrices. In: Apostolico, A., Galil, Z. (eds.) Pattern Matching Algorithms, ch. 11,, pp. 293–340. Oxford University Press, Oxford (1997)
6. Giancarlo, R., Guaiana, D.: On-line construction of two-dimensional suffix trees. J. Complex. **15**, 72–127 (1999)
7. Gonnet, G.H.: Efficient searching of text and pictures. Tech. Report OED-88-02, University of Waterloo (1988)
8. Kim, D.K., Kim, Y.A., Park, K.: Constructing suffix arrays for multidimensional matrices. In: Proceedings of the 9th Symposium on Combinatorial Pattern Matching, 1998, pp. 249–260
9. Kim, D.K., Kim, Y.A., Park, K.: Generalizations of suffix arrays to multi-dimensional matrices. Theor. Comput. Sci. **302**, 401–416 (2003)
10. Kim, D.K., Park, K.: Linear-time construction of two-dimensional suffix trees. In: Proceedings of the 26th International Colloquium on Automata, Languages, and Programming, 1999, pp. 463–372
11. Manber, U., Myers, G.: Suffix arrays: A new method for on-line string searches. SIAM J. Comput. **22**, 935–948 (1993)
12. McCreight, E.M.: A space-economical suffix tree construction algorithms. J. ACM **23**, 262–272 (1976)
13. Na, J.C., Giancarlo, R., Park, K.: $O(n^2 \log n)$ time on-line construction of two-dimensional suffix trees. In: Proceedings of the 11th International Computing and Combinatorics Conference, 2005, pp. 273–282
14. Storer, J.A.: Lossless image compression using generalized LZ1-type methods. In: Proceedings of Data Compression Conference, 1996, pp. 290–299

Two-Dimensional Pattern Matching with Scaling

▶ Two-Dimensional Scaled Pattern Matching

Two-Dimensional Scaled Pattern Matching

2006; Amir, Chencinski

AMIHOOD AMIR[1]

[1] Department of Computer Science, Bar-Ilan University, Ramat-Gan, Israel

[2] Department of Computer Science, Johns Hopkins University, Baltimore, MD, USA

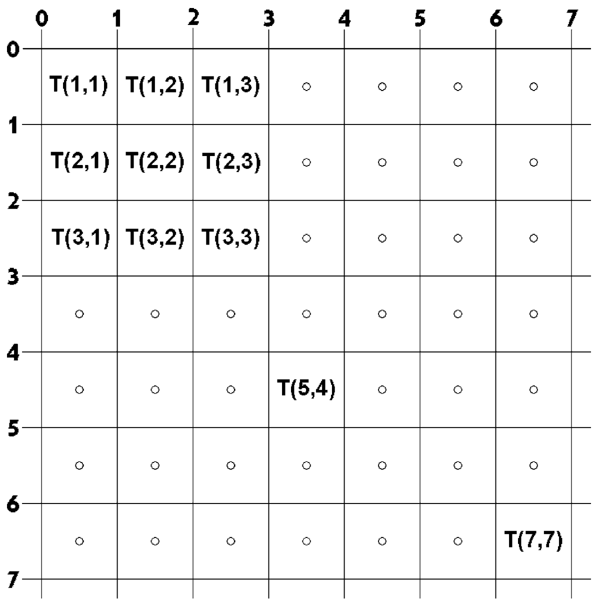
Keywords and Synonyms

Pattern matching in scaled images; 2d scaled matching; Two dimensional pattern matching with scaling; Multidimensional scaled search

Problem Definition

Definition 1 Let T be a two-dimensional $n \times n$ array over some alphabet Σ.

1. The *unit pixels array* for T (T^{1X}) consists of n^2 unit squares, called pixels in the real plane $\Re^2$. The corners of the pixel $T[i, j]$ are $(i-1, j-1)$, $(i, j-1)$, $(i-1, j)$, and (i, j). Hence the pixels of T form a regular $n \times n$ array that covers the area between $(0, 0)$, $(n, 0)$, $(0, n)$, and (n, n). Point $(0, 0)$ is the *origin* of the unit pixel array. The *center* of each pixel is the geometric center point of its square location. Each pixel $T[i, j]$ is identified with the value from Σ that the original array T had in that position. Say that the pixel has a color or a character from Σ. See Fig. 1 for an example of the grid and pixel centers of a 7×7 array.
2. Let $r \in \Re$, $r \geq 1$. The *r-ary pixels array* for T (T^{rX}) consists of n^2 *r-squares*, each of dimension $r \times r$ whose *origin* is $(0, 0)$ and covers the area between $(0, 0)$, $(nr, 0)$, $(0, nr)$, and (nr, nr). The corners of the pixel $T[i, j]$ are $((i-1)r, (j-1)r)$, $(ir, (j-1)r)$, $((i-1)r, jr)$, and (ir, jr). The *center* of each pixel is the geometric center point of its square location.



Two-Dimensional Scaled Pattern Matching, Figure 1
The grid and pixel centers of a unit pixel array for a 7 × 7 array

Two-Dimensional Scaled Pattern Matching, Figure 2
An original image, scaled by 1.3 and scaled by 2, using the geometric model definition of scaling

Notation: Let $r \in \Re$. $[r]$ denotes the *rounding* of r, i. e.

$$[r] = \begin{cases} \lfloor r \rfloor & \text{if } r - \lfloor r \rfloor < .5; \\ \lceil r \rceil & \text{otherwise.} \end{cases}$$

Definition 2 Let T be an $n \times n$ text array, P be an $m \times m$ pattern array over alphabet Σ, and let $r \in \Re$, $1 \le r \le \frac{n}{m}$. Say that there is an *occurrence of P scaled to r* at text location (i, j) if the following conditions hold:

Let T^{1X} be the unit pixels array of T and P^{rX} be the r-ary pixel arrays of P. Translate P^{rX} onto T^{1X} in a manner that the origin of P^{rX} coincides with location $(i-1, j-1)$ of T^{1X}. Every center of a pixel in T^{1X} which is within the area covered by $(i-1, j-1), (i-1, j-1+mr), (i-1+mr, j-1)$ and $(i-1+mr, j-1+mr)$ has the same color as the r-square of P^{rX} in which it falls.

The colors of the centers of the pixels in T^{1X} which are within the area covered by $(i-1, j-1), (i-1, j-1+mr), (i-1+mr, j-1)$ and $(i-1+mr, j-1+mr)$ define a $[mr] \times [mr]$ array over Σ. This array is denoted by $P^{s(r)}$ and called P *scaled to r*.

The above definition is the one provided in the *geometric model*, pioneered by Landau and Vishkin [15], and Fredriksson and Ukkonen [14]. Prior to the advent of the geometric model, the only discrete definition of scaling was to natural scales, as defined by Amir, Landau and Vishkin [10]:

Definition 3 Let $P[m \times m]$ be a two-dimensional matrix over alphabet Σ (not necessarily bounded). Then P scaled by s (P^s) is the $sm \times sm$ matrix where every symbol $P[i, j]$ of P is replaced by a $s \times s$ matrix whose elements all equal the symbol in $P[i, j]$. More precisely,

$$P^s[i, j] = P[\lceil \frac{i}{s} \rceil, \lceil \frac{j}{s} \rceil] .$$

Say that pattern $P[m \times m]$ *occurs* (or an occurrence of P starts) at location (k, l) of the text T if for any $i \in \{1, \dots, m\}$ and $j \in \{1, \dots, m\}$, $T[k+i-1, l+j-1] = P[i, j]$.

The *two dimensional pattern matching problem with natural scales* is defined as follows.

INPUT: Pattern matrix $P[i, j]$ $i = 1, \dots m; j = 1, \dots, m$ and Text matrix $T[i, j]$ $i = 1, \dots, n; j = 1, \dots, n$ where $n > m$.

OUTPUT: all locations in T where an occurrence of P scaled by s (an s-occurrence) starts, for any $s = 1, \dots, \lfloor \frac{n}{m} \rfloor$.

The natural scales definition cannot answer normal everyday occurrences such as an image scaled to, say, 1.3. This led to the geometric model. The geometric model is a discrete adaptation, without smoothing, of scaling as used in computer graphics. The definition is pleasing in a "real-world" sense. Figure 2 shows "lenna" scaled to non-discrete scales by the geometric model definition. The results look natural.

It is possible, of course, to consider a *one dimensional* version of scaling, or scaling in *strings*. Both above definitions apply for one dimensional scaling where the text and pattern are taken to be matrices having a single row. The interest in one dimensional scaling lies because of two reasons: (1) There is a faster algorithm for one dimensional scaling in the geometric model than the restriction of the two dimensional scaling algorithm to one dimension. (2) Historically, before the geometric model was defined, there was an attempt [3] to define real scaling on strings as follows.

Definition 4 Denote the string $aa \cdots a$, where a is repeated r times, by a^r. The *one dimensional floor real scaled matching problem* is the following.

INPUT: A pattern $P = a_1^{r_1} a_2^{r_2} \dots a_j^{r_j}$, of length m, and a text T of length n.
OUTPUT: All locations in the text where the substring $a_1^{c_1} a_2^{\lfloor r_2 k \rfloor} \dots a_{j-1}^{\lfloor r_{j-1} k \rfloor} a_j^{c_j}$ appears, where $c_1 \geq \lfloor r_1 k \rfloor$ and $c_j \geq \lfloor r_j k \rfloor$.

This definition indeed handles real scaling but has a significant weakness in that a string of length m scaled to r may be significantly shorter than mr. For this reason the definition could not be generalized to two dimensions. The geometric model does not suffer from these deficiencies.

Key Results

The first results in scaled natural matching dealt with fixed finite-sized alphabets.

Theorem 1 (Amir, Landau, and Vishkin 1992 [10]) *There exists an $O(|T| \log |\Sigma|)$ worst-case time solution to the two dimensional pattern matching problem with natural scales, for fixed finite alphabet Σ.*

The main idea behind the algorithm is analyzing the text with the aid of *power columns*. Those are the text columns appearing $m - 1$ columns apart, where P is an $m \times m$ pattern. This dependence on the pattern size make the power columns useless where a dictionary of different sized patterns is involved. A significantly simpler algorithm with an additional advantage of being alphabet-independent was presented in [6].

Theorem 2 (Amir and Calinescu 2000 [6]) *There exists an $O(|T|)$ worst-case time solution to the two dimensional pattern matching problem with natural scales.*

The alphabet independent time complexity of this algorithm was achieved by developing a scaling-invariant "signature" of the pattern. This idea was further developed to scaled dictionary matching.

Theorem 3 (Amir and Calinescu 2000 [6]) *Given a static dictionary of square pattern matrices. It is possible in $O(|D| \log k)$ preprocessing, where $|D|$ is the total dictionary size and k is the number of patterns in the dictionary, and $O(|T| \log k)$ text scanning time, for input text T, to find all occurrences of dictionary patterns in the text in all natural scales.*

This is identical to the time at [8], the best non-scaled matching algorithm for a static dictionary of square patterns. It is somewhat surprising that scaling does not add to the complexity of single matching nor dictionary matching.

The first algorithm to solve the scaled matching problem for real scales, was a one dimensional real scaling algorithm using Definition 4.

Theorem 4 (Amir, Butman, and Lewenstein 1998 [3]) *There exists an $O(|T|)$ worst-case time solution to the one dimensional floor real scaled matching problem.*

The first algorithm to solve the two dimensional scaled matching problem for real scales in the geometric model is the following.

Theorem 5 (Amir, Butman, Lewenstein, and Porat 2003 [4]) *Given an $n \times n$ text and $m \times m$ pattern. It is possible to find all pattern occurrences in all real scales in time $O(nm^3 + n^2 m \log m)$ and space $O(nm^3 + n^2)$.*

The above result was improved.

Theorem 6 (Amir and Chencinski 2006 [7]) *Given an $n \times n$ text and $m \times m$ pattern. It is possible to find all pattern occurrences in all real scales in time $O(n^2 m)$ and space $O(n^2)$.*

This algorithm achieves its time by exploiting geometric characteristics of nested scales occurrences and a sophisticated use of dueling [1,16].

The assumption in both above algorithms is that the scaled occurrence of the pattern starts at the top left corner of some pixel.

It turns out that one can achieve faster times in the *one dimensional* real scaled matching problem, even in the geometric model.

Theorem 7 (Amir, Butman, Lewenstein, Porat, and Tsur 2004 [5]) *Given a text string T of length n and a pattern string P of length m, there exists an $O(n \log m + m\sqrt{nm \log m})$ worst-case time solution to the one dimensional pattern matching problem with real scales in the geometric model.*

Applications

The problem of finding approximate occurrences of a template in an image is a central one in digital libraries and web searching. The current algorithms to solve this problem use methods of computer vision and computational geometry. They model the image in another space and seek a solution there. A deterministic worst-case algorithm in pixel-level images does not yet exist. Yet, such an algorithm could be useful, especially in raw data that has not been modeled, e. g. movies. The work described here advances another step toward this goal from the scaling point of view.

Open Problems

Finding all scaled occurrences without fixing the scaled pattern start at the top left corner of the text pixel would be important from a practical point of view. The final goal is an integration of scaling with rotation [2,11,12,13] and local errors (edit distance) [9].

Cross References

▶ Multidimensional Compressed Pattern Matching
▶ Multidimensional String Matching

Recommended Reading

1. Amir, A., Benson, G., Farach, M.: An alphabet independent approach to two dimensional pattern matching. SIAM J. Comput. **23**(2), 313–323 (1994)
2. Amir, A., Butman, A., Crochemore, M., Landau, G.M., Schaps, M.: Two-dimensional pattern matching with rotations. Theor. Comput. Sci. **314**(1–2), 173–187 (2004)
3. Amir, A., Butman, A., Lewenstein, M.: Real scaled matching. Inf. Proc. Lett. **70**(4), 185–190 (1999)
4. Amir, A., Butman, A., Lewenstein, M., Porat, E.: Real two dimensional scaled matching. In: Proc. 8th Workshop on Algorithms and Data Structures (WADS '03), pp. 353–364 (2003)
5. Amir, A., Butman, A., Lewenstein, M., Porat, E., Tsur, D.: Efficient one dimensional real scaled matching. In: Proc. 11th Symposium on String Processing and Information Retrieval (SPIRE '04), pp. 1–9 (2004)
6. Amir, A., Calinescu, G.: Alphabet independent and dictionary scaled matching. J. Algorithms **36**, 34–62 (2000)
7. Amir, A., Chencinski, E.: Faster two dimensional scaled matching. In: Proc. 17th Symposium on Combinatorial Pattern Matching (CPM). LNCS, vol. 4009, pp. 200–210. Springer, Berlin (2006)
8. Amir, A., Farach, M.: Two dimensional dictionary matching. Inf. Proc. Lett. **44**, 233–239 (1992)
9. Amir, A., Landau, G.: Fast parallel and serial multidimensional approximate array matching. Theor. Comput. Sci. **81**, 97–115 (1991)
10. Amir, A., Landau, G.M., Vishkin, U.: Efficient pattern matching with scaling. J. Algorithms **13**(1), 2–32 (1992)
11. Amir, A., Tsur, D., Kapah, O.: Faster two dimensional pattern matching with rotations. In: Proc. 15th Annual Symposium on Combinatorial Pattern Matching (CPM '04), pp. 409–419 (2004)
12. Fredriksson, K., Mäkinen, V., Navarro, G.: Rotation and lighting invariant template matching. In: Proceedings of the 6th Latin American Symposium on Theoretical Informatics (LATIN'04). LNCS, pp. 39–48 (2004)
13. Fredriksson, K., Navarro, G., Ukkonen, E.: Optimal exact and fast approximate two dimensional pattern matching allowing rotations. In: Proceedings of the 13th Annual Symposium on Combinatorial Pattern Matching (CPM 2002). LNCS, vol. 2373, pp. 235–248 (2002)
14. Fredriksson, K., Ukkonen, E.: A rotation invariant filter for two-dimensional string matching. In: Proc. 9th Annual Symposium on Combinatorial Pattern Matching (CPM). LNCS, vol. 1448, pp. 118–125. Springer, Berlin (1998)
15. Landau, G.M., Vishkin, U.: Pattern matching in a digitized image. Algorithmica **12**(3/4), 375–408 (1994)
16. Vishkin, U.: Optimal parallel pattern matching in strings. Proc. 12th ICALP, pp. 91–113 (1985)

Two-Interval Pattern Problems

2004; Vialette
2007; Cheng, Yang, Yuan

STÉPHANE VIALETTE
IGM-LabInfo, University of Paris-East, Descartes, France

Keywords and Synonyms

2-intervals; RNA structures

Problem Definition

The problem is concerned with finding large constrained patterns in sets of 2-intervals. Given a single-stranded RNA molecule, a sequence of contiguous bases of the molecule can be represented as an interval on a single line, and a possible pairing between two disjoint sequences can be represented as a 2-interval, which is merely the union of two disjoint intervals. Derived from arc-annotated sequences, 2-interval representation considers thus only the bonds between the bases and the pattern of the bonds, such as hairpin structures, knots and pseudoknots. A maximum cardinality disjoint subset of a candidate set of 2-intervals restricted to certain prespecified geometrical constraints can provide a useful valid approximation for RNA secondary structure determination.

The geometric properties of 2-intervals provide a possible guide for understanding the computational complexity of finding structured patterns in RNA sequences. Using a model to represent nonsequential information allows us to vary restrictions on the complexity of the pattern structure. Indeed, two disjoint 2-intervals, i. e., two 2-intervals that do not intersect in any point, can be in precedence order ($<$), be allowed to nest ($\sqsubset$) or be allowed to cross ($\between$). Furthermore, the set of 2-intervals and the pattern can have different restrictions, e. g., all intervals have the same length or all the intervals are disjoint. These different combinations of restrictions alter the computational complexity of the problems, and need to be examined separately. This examination produces efficient algorithms for more restrictive structured patterns, and hardness results for those that are less restrictive.

Notations

Let $I = [a, b]$ be an interval on the line. Write $\mathsf{start}(I) = a$ and $\mathsf{end}(I) = b$. A *2-interval* is the union of two disjoint intervals defined over a single line and is denoted by $D = (I, J)$; I is completely to the left of J. Write $\mathsf{left}(D) = I$ and $\mathsf{right}(D) = J$. Two 2-intervals $D_1 = (I_1, J_1)$ and $D_2 = (I_2, J_2)$ are said to be *disjoint* (or *nonintersecting*) if both 2-intervals share no common point, i. e., $(I_1 \cup J_1) \cap (I_2 \cup J_2) = \emptyset$. For such disjoint pairs of 2-intervals, three natural binary relations, denoted $<$, $\sqsubset$ and $\between$ are of special interest:

- $D_1 < D_2$ (D_1 *precedes* D_2), if $I_1 < J_1 < I_2 < J_2$,
- $D_1 \sqsubset D_2$ (D_1 is *nested in* D_2), if $I_2 < I_1 < J_1 < J_2$, and
- $D_1 \between D_2$ (D_1 *crosses* D_2), if $I_1 < I_2 < J_1 < J_2$.

A pair of 2-intervals D_1 and D_2 is said to be *R-comparable* for some $R \in \{<, \sqsubset, \between\}$, if either $D_1 R D_2$ or $D_2 R D_1$. Note that any two disjoint 2-intervals are R-comparable for some $R \in \{<, \sqsubset, \between\}$. A set of disjoint 2-intervals $\mathcal{D}$ is said to be $\mathcal{R}$*-comparable* for some $\mathcal{R} \subseteq \{<, \sqsubset, \between\}$, $\mathcal{R} \neq \emptyset$, if any pair of distinct 2-intervals in $\mathcal{D}$ is R-comparable for some $R \in \mathcal{R}$. The nonempty subset $\mathcal{R}$ is called a *model* for $\mathcal{D}$.

The 2-interval-pattern problem asks one to find in a set of 2-intervals a largest subset of pairwise compatible 2-intervals. In the present context, compatibility denotes the fact that any two 2-intervals in the solution are (1) nonintersecting and (2) satisfy some prespecified geometrical constraints. The 2-interval-pattern problem is formally defined as follows:

Problem 1 (2-interval-pattern)
INPUT: *A set of 2-intervals $\mathcal{D}$ and a model $\mathcal{R} \subseteq \{<, \sqsubset, \between\}$.*
SOLUTION: *A $\mathcal{R}$-comparable subset $\mathcal{D}' \subseteq \mathcal{D}$.*
MEASURE: *The size of the solution, i. e., $|\mathcal{D}'|$.*

According to the above definition, any solution for the 2-interval-pattern problem for some model $\mathcal{R} \subseteq \{<, \sqsubset, \between\}$ corresponds to an RNA structure constrained by $\mathcal{R}$. For example, a solution for the 2-interval-pattern problem for the $\mathcal{R} = \{<, \sqsubset\}$ model corresponds to a pseudoknot-free structure (a *pseudoknot* in an RNA sequence $S = s_1, s_2, \dots, s_n$ is composed of two interleaving nucleotide pairings (s_i, s_j) and $(s_{i'}, s_{j'})$ such that $i < i' < j < j'$).

Some additional definitions are needed for further algorithmic analysis. Let $\mathcal{D}$ be a set of 2-intervals. The *width* (respectively *height*, *depth*) is the size of a maximum cardinality $\{<\}$-comparable (respectively $\{\sqsubset\}$-comparable, $\{\between\}$-comparable) subset $\mathcal{D}' \subseteq \mathcal{D}$. The *interleaving distance* of a 2-interval $D_i \in \mathcal{D}$ is defined to be the distance between the two intervals of D_i, i. e., $\mathsf{start}(\mathsf{right}(D_i)) - \mathsf{end}(\mathsf{left}(D_i))$. The *total interleaving distance* of the set of 2-intervals $\mathcal{D}$, written $\mathcal{L}(\mathcal{D})$, is the sum of all interleaving distances, i. e., $\mathcal{L}(\mathcal{D}) = \sum_{D_i \in \mathcal{D}} \mathsf{start}(\mathsf{right}(D_i)) - \mathsf{end}(\mathsf{left}(D_i))$. The *interesting coordinates* of $\mathcal{D}$ are defined to be the set $X(\mathcal{D}) = \bigcup_{D_i \in \mathcal{D}} \{\mathsf{end}(\mathsf{left}(D_i)), \mathsf{start}(\mathsf{right}(D_i))\}$. The *density* of $\mathcal{D}$, written $d(\mathcal{D})$, is the maximum number of 2-intervals in $\mathcal{D}$ over a single point. Formally, $d(\mathcal{D}) = \max_{x \in X(\mathcal{D})} \{D \in \mathcal{D} : \mathsf{end}(\mathsf{left}(D) \leq x < \mathsf{start}(\mathsf{right}(D))\}$.

Constraints

The structure of the set of all (simple) intervals involved in a set of 2-intervals $\mathcal{D}$ turns out to be of particular importance for algorithmic analysis of the 2-interval-pattern problem. The *interval ground set* of $\mathcal{D}$, denoted $\mathcal{I}(\mathcal{D})$, is the set of all intervals involved in $\mathcal{D}$, i. e., $\mathcal{I}(\mathcal{D}) = \{\mathsf{left}(D_i) : D_i \in \mathcal{D}\} \cup \{\mathsf{right} < (D_i) : D_i \in \mathcal{D}\}$. In [7,20], four types of interval ground sets were introduced.

1. *Unlimited*: no restriction on the structure.
2. *Balanced*: each 2-interval $D_i \in \mathcal{D}$ is composed of two intervals having the same length, i. e., $|\mathsf{left}(D_i)| = |\mathsf{right}(D_i)|$.
3. *Unit*: the interval ground set $\mathcal{I}(\mathcal{D})$ is solely composed of unit length intervals.
4. *Disjoint*: no two distinct intervals in the interval ground set $\mathcal{I}(\mathcal{D})$ intersect.

Observe that a unit 2-interval set is balanced, while the converse is not necessarily true. Furthermore, for most applications, one may assume that a disjoint 2-interval set is unit. Observe that in this latter case, a set of 2-intervals reduces to a graph $G = (V, E)$ equipped with a numbering of its vertices from 1 to $|V|$, and hence the 2-interval-pattern problem for disjoint interval ground sets reduces to finding a constrained maximum matching in a linear graph. Considering additional restrictions such as:

- Bounding the width, the height or the depth of either the input set of 2-intervals or the solution subset
- Bounding the interleaving distances

is also of interest for practical applications.

Key Results

The different combinations of the models and interval ground sets alter the computational complexity of the 2-interval-pattern problem. The main results are summarized in Tables 1 (time complexity and hardness) and 2 (approximation for hard instances).

Theorem 1 *The 2-interval-pattern problem is approximable (APX) hard for models $\mathcal{R} = \{<, \sqsubset, \between\}$ and $\mathcal{R} = \{\sqsubset, \between\}$, and is nondeterministic polynomial-time (NP) complete – in its natural decision version – for model $\mathcal{R} = \{<, \between\}$, even when restricted to unit interval ground sets.*

Two-Interval Pattern Problems, Table 1

Complexity of the 2-interval-pattern problem for all combinations of models and interval ground sets. For the polynomial-time cases, $n = |\mathcal{D}|$, $\mathcal{L} = \mathcal{L}(\mathcal{D})$ and $d = d(\mathcal{D})$

Model $\mathcal{R}$	Interval ground set $\mathcal{I}(\mathcal{D})$	
	Unlimited, Balanced, Unit	Disjoint
$\{<,\sqsubset,\between\}$	APX-hard [1]	$O(n\sqrt{n})$ [15]
$\{<,\between\}$	NP-complete [3]	**unknown**
$\{\sqsubset,\between\}$	APX-hard [19]	$O(n\log n + \mathcal{L})$ [8]
$\{<,\sqsubset\}$	$O(n\log n + nd)$ [8]	
$\{<\}$	$O(n\log n)$ [19]	
$\{\sqsubset\}$	$O(n\log n)$ [3]	
$\{\between\}$	$O(n\log n + \mathcal{L})$ [8]	

Notice here that the 2-interval-pattern problem for model $\mathcal{R} = \{<,\between\}$ is not APX-hard. Two hard cases of the 2-interval-pattern turn out to be polynomial-time-solvable when restricted to disjoint-interval ground sets.

Theorem 2 *The 2-interval-pattern problem for a disjoint-interval ground set is solvable in*

- *$O(n\sqrt{n})$ time for model $\mathcal{R} = \{<,\sqsubset,\between\}$ (trivial reduction to the standard maximum matching problem)*
- *$O(n\log n + \mathcal{L})$ time for model $\mathcal{R} = \{\sqsubset,\between\}$*

The complexity of the 2-interval-pattern problem for model $\mathcal{R} = \{<,\between\}$ and a disjoint-interval ground set is still unknown. Three cases of the 2-interval-pattern problem are polynomial-time-solvable, regardless of the structure of the interval ground sets.

Theorem 3 *The 2-interval-pattern problem is solvable in*

- *$O(n\log n + nd)$ time for model $\mathcal{R} = \{<,\sqsubset\}$*
- *$O(n\log n)$ time for models $\mathcal{R} = \{<\}$ and $\mathcal{R} = \{\sqsubset\}$*
- *$O(n\log n + \mathcal{L})$ time for model $\mathcal{R} = \{\between\}$*

One may now turn to approximating hard instances of the 2-interval-pattern problem. Surprisingly enough, no significant differences (in terms of approximation guarantees) have yet been found for the 2-interval-pattern problem between the model $\mathcal{R} = \{<,\sqsubset,\between\}$ and the model $\mathcal{R} = \{\sqsubset,\between\}$ (the approximation algorithms are, however, different).

Theorem 4 *The 2-interval-pattern problem for model $\mathcal{R} = \{<,\sqsubset,\between\}$ or model $\mathcal{R} = \{\sqsubset,\between\}$ is approximable within ratio*

- *4 for unlimited-interval ground sets, and*
- *$2 + \epsilon$ for unit-interval ground sets.*

The 2-interval-pattern problem for model $\mathcal{R} = \{<,\between\}$ is approximable within ratio $1 + 1/\epsilon$, $\epsilon \geq 2$ for all models.

A practical 3-approximation algorithm for model $\mathcal{R} = \{<,\sqsubset,\between\}$ (resp. $\mathcal{R} = \{\sqsubset,\between\}$) and unit interval ground set that runs in $\mathcal{O}(n\lg n)$ (resp. $\mathcal{O}(n^2\lg n)$) time has been proposed in [1] (resp. [7]). For model $\mathcal{R} = \{<,\between\}$, a more practical 2-approximation algorithm that runs in $\mathcal{O}(n^3\lg n)$ time has been proposed in [10]. Notice that Theorem 4 holds true for the weighted version of the 2-interval-pattern problem [7] except for models $\mathcal{R} = \{<,\sqsubset,\between\}$ and $\mathcal{R} = \{\sqsubset,\between\}$ and unit interval ground set where the best approximation ratio is $2.5 + \epsilon$ [5].

Applications

Sets of 2-intervals can be used for modeling stems in RNA structures [20,21], determining DNA sequence similarities [13] or scheduling jobs that are given as groups of non-intersecting segments in the real line [1,9]. In all these applications, one is concerned with finding a maximum cardinality subset of nonintersecting 2-intervals. Some other classical combinatorial problems are also of interest [5]. Also, considering sets of t-intervals (each element is the union of at most t disjoint intervals) and their corresponding intersection graph has proved to be useful.

It is computationally challenging to predict RNA structures including pseudoknots [14]. Practical approaches to cope with intractability are either to restrict the class of pseudoknots under consideration [18] or to use heuristics [6,17,19]. The general problem of establishing a general representation of structured patterns, i. e., *macroscopic describers* of RNA structures, was considered in [20]. Sets of 2-intervals provide such a natural geometric description.

Constructing a relevant 2-interval set from a RNA sequence is relatively easy: stable stems are selected, usually according to a simplified thermodynamic model without accounting for loop energy [2,16,19,20,21]. Predicting a reliable RNA structure next reduces to finding a maximum subset of nonconflicting 2-intervals, i. e., a subset of disjoint 2-intervals. Considering in addition a model $\mathcal{R} \subseteq \{<,\sqsubset,\between\}$ allows us to vary restrictions on the complexity of the pattern structure. In [21], the treewidth of the intersection graph of the set of 2-intervals is considered for speeding up the computation.

For sets of 2-intervals involved in practical applications, restrictions on the interval ground set are needed. Unit interval ground sets were considered in [7]. Of particular importance in the context of molecular biology (RNA structures and DNA sequence similarities) are balanced interval ground sets, where each 2-interval is composed of two equally length intervals.

Two-Interval Pattern Problems, Table 2
Performance ratios for hard instances of the 2-interval-pattern problem. LP stands for *Linear Programming* and N/A stands for *Not Applicable*

Model $\mathcal{R}$	Interval ground set $\mathcal{I}(\mathcal{D})$			
	Unlimited	Balanced	Unit	Disjoint
$\{<, \sqsubset, \between\}$	4 LP [1]	4 $\mathcal{O}(n \lg n)$ [7]	$2 + \epsilon$ $\mathcal{O}(n^2 + n^{\mathcal{O}(\log 1/\epsilon)})$ [13]	N/A
$\{\sqsubset, \between\}$	4 LP [7]	4 $\mathcal{O}(n^2 \lg n)$ [7]	$2 + \epsilon$ $\mathcal{O}(n^2 + n^{\mathcal{O}(\log 1/\epsilon)})$ [13]	N/A
$\{<, \between\}$	$1 + 1/\epsilon$ $\quad \mathcal{O}(n^{2\epsilon+3}),\ \epsilon \geq 2$ [14]			

Open Problems

A number of problems related to the 2-interval-pattern problem remain open. First, improving the approximation ratios for the various flavors of the 2-interval-pattern problem is of particular importance. For example, the existence of a fast approximation algorithm with good performance guarantee for the 2-interval-pattern problem for model $\mathcal{R} = \{<, \sqsubset, \between\}$ remains an apparently challenging open problem. A related open research area is concerned with balanced-interval ground sets. In particular, no evidence has shown yet that the 2-interval-pattern problem becomes easier to approximate for balanced-interval ground sets. This question is of special importance in the context of RNA structures where most 2-intervals are balanced.

A number of important question are still open for model $\mathcal{R} = \{<, \between\}$. First, it is still unknown whether the 2-interval-pattern problem for disjoint-interval ground sets and model $\mathcal{R} = \{<, \between\}$ is polynomial-time-solvable. Observe that this problem trivially reduces to the following graph problem: Given a graph $G = (V, E)$ with $V = \{1, 2, \ldots, n\}$, find a maximum cardinality matching $\mathcal{M} \subseteq E$ such that for any two distinct edges $\{i, j\}$ and $\{k, l\}$ of $\mathcal{M}$, $i < j$, $k < l$ and $i < k$, either $j < k$ or $j < l$. Another open question concerns the approximation of the 2-interval-pattern problem for balanced interval ground set. Is this special case better approximable than the general case?

A last direction of research is concerned with the parameterized complexity of the 2-interval-pattern problem. For example, it is not known whether the 2-interval-pattern problem for models $\mathcal{R} = \{<, \sqsubset, \between\}$, $\mathcal{R} = \{\sqsubset, \between\}$ or $\mathcal{R} = \{<, \between\}$ is fixed-parameter-tractable when parameterized by the size of the solution. Also, investigating the parameterized complexity for parameters such as the maximum number of pairwise crossing intervals in the input set or the treewidth of the corresponding intersection 2-interval graph, which are expected to be relatively small for most practical applications, is of particular interest.

Cross References

▶ RNA Secondary Structure Prediction Including Pseudoknots
▶ RNA Secondary Structure Prediction by Minimum Free Energy

Recommended Reading

1. Bar-Yehuda, R., Halldorsson, M., Naor, J., Shachnai, H., Shapira, I.: Scheduling split intervals. In: Proc. 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 2002, pp. 732–741
2. Billoud, B., Kontic, M., Viari, A.: Palingol a declarative programming language to describe nucleic acids' secondary structures and to scan sequence database. Nucleic. Acids. Res. **24**, 1395–1403 (1996)
3. Blin, G., Fertin, G., Vialette, S.: Extracting 2-intervals subsets from 2-interval sets. Theor. Comput. Sci. **385**(1–3), 241–263 (2007)
4. Blin, G., Fertin, G., Vialette, S.: New results for the 2-interval pattern problem. In: Proc. 15th Annual Symposium on Combinatorial Pattern Matching (CPM). Lecture Notes in Computer Science, vol. 3109. Springer, Berlin (2004)
5. Butman, A., Hermelin, D., Lewenstein, M., Rawitz, D.: Optimization problems in multiple-interval graphs. In: Proc. 9th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), ACM-SIAM, 2007, pp. 268–277
6. Chen, J.-H., Le, S.-Y., Maize, J.: Prediction of common secondary structures of RNAs: a genetic algorithm approach. Nucleic. Acids. Res. **28**, 991–999 (2000)
7. Crochemore, M., Hermelin, D., Landau, G., Rawitz, D., Vialette, S.: Approximating the 2-interval pattern problem, Theoretical Computer Science (special issue for Alberto Apostolico) (2008)
8. Erdong, C., Linji, Y., Hao, Y.: Improved algorithms for 2-interval pattern problem. J. Combin. Optim. **13**(3), 263–275 (2007)
9. Halldorsson, M., Karlsson, R.: Strip graphs: Recognition and scheduling. In: Proc. 32nd International Workshop on Graph-Theoretic Concepts in Computer Science (WG). Lecture Notes in Computer Science, vol. 4271, pp. 137–146. Springer, Berlin (2006)
10. Jiang, M.: A 2-approximation for the preceding-and-crossing structured 2-interval pattern problem, J. Combin. Optim. **13**, 217–221 (2007)
11. Jiang, M.: Improved approximation algorithms for predicting RNA secondary structures with arbitrary pseudoknots. In: Proc. 3rd International Conference on Algorithmic Aspects in Infor-

mation and Management (AAIM), Portland, OR, USA, Lecture Notes in Computer Science, vol. 4508, pp. 399–410. Springer (2007)

12. Jiang, M.: A PTAS for the weighted 2-interval pattern problem over the preceding-and-crossing model. In: Y.X. A.W.M. Dress, B. Zhu (eds.) Proc. 1st Annual International Conference on Combinatorial Optimization and Applications (COCOA), Xi'an, China, Lecture Notes in Computer Science, vol. 4616, pp. 378–387. Springer (2007)
13. Joseph, D., Meidanis, J., Tiwari, P.: Determining DNA sequence similarity using maximum independent set algorithms for interval graphs. In: Proc. 3rd Scandinavian Workshop on Algorithm Theory (SWAT). Lecture Notes in Computer Science, pp. 326–337. Springer, Berlin (1992)
14. Lyngsø, R., Pedersen, C.: RNA pseudoknot prediction in energy-based models. J. Comput. Biol. **7**, 409–427 (2000)
15. Micali, S., Vazirani, V.. An $O(sqrt|V||E|)$ algorithm for finding maximum matching in general graphs. In: Proc. 21st Annual Symposium on Foundation of Computer Science (FOCS), IEEE, 1980, pp. 17–27
16. Nussinov, R., Pieczenik, G., Griggs, J., Kleitman, D.: Algorithms for loop matchings. SIAM J. Appl. Math. **35**, 68–82 (1978)
17. Ren, J., Rastegart, B., Condon, A., Hoos, H.: HotKnots: Heuristic prediction of rna secondary structure including pseudoknots. RNA **11**, 1194–1504 (2005)
18. Rivas, E., Eddy, S.: A dynamic programming algorithm for RNA structure prediction including pseudoknots. J. Mol. Biol. **285**, 2053–2068 (1999)
19. Ruan, J., Stormo, G., Zhang, W.: An iterated loop matching approach to the prediction of RNA secondary structures with pseudoknots. Bioinformatics **20**, 58–66 (2004)
20. Vialette, S.: On the computational complexity of 2-interval pattern matching. Theor. Comput. Sci. **312**, 223–249 (2004)
21. Zhao, J., Malmberg, R., Cai, L.: Rapid ab initio rna folding including pseudoknots via graph tree decomposition. In: Proc. Workshop on Algorithms in Bioinformatics. Lecture Notes in Computer Science, vol. 4175, pp. 262–273. Springer, Berlin (2006)

Two-Level Boolean Minimization

1956; McCluskey

ROBERT DICK
Department Electrical Engineering and Computer Systems, Northwestern University, Evanston, IL, USA

Keywords and Synonyms

Logic minimization; Quine–McCluskey algorithm; Tabular method

Problem Definition

Summary

Find a minimal sum-of-products expression for a Boolean function.

Two-Level Boolean Minimization, Table 1
Equivalent representations with different implementation complexities

Expression	Meaning in english	Boolean logic identity
$\overline{a} \wedge \overline{b} \vee \overline{a} \wedge b$	not *a* and not *b* or not *a* and *b*	Distributivity
$\overline{a} \wedge (\overline{b} \vee b)$	not *a* and either not *b* or *b*	Complements
$\overline{a} \wedge True$	not *a* and *True*	Boundedness
$\overline{a}$	not *a*	

Extended Definition

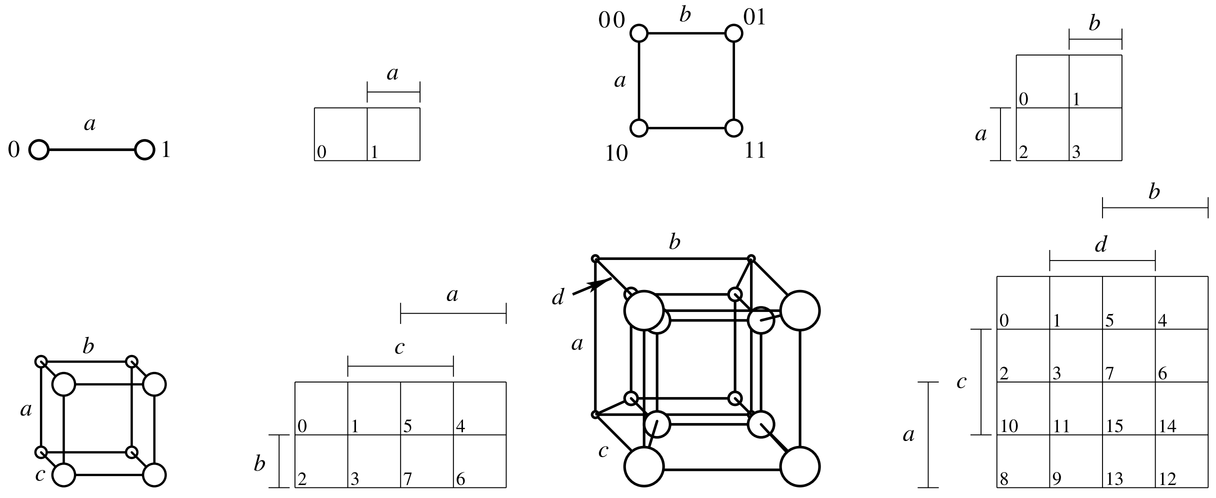
Consider a Boolean algebra with two elements: *False* or *True*. A Boolean function $f(y_1, y_2, \cdots, y_n)$ of n Boolean input variables specifies an output value for each combination of input variable values. It is possible to represent the same function with a number of different expressions. For example, the first and last expressions in Table 1 correspond to this function. Assuming access to complemented input variables, straight-forward implementations of these expressions would require two *and* gates and an *or* gate for $\overline{a} \wedge \overline{b} \vee \overline{a} \wedge b$ and only a wire for $\overline{a}$. Although the implementation efficiency depends on target technology, in general terser expressions enable greater efficiency. Boolean minimization is the task of deriving the tersest expression for a function. Elegant and optimal algorithms exist for solving the variant of this problem in which the expression is limited to two levels, i. e., a layer of *and* gates followed by a single *or* gate or a layer of *or* gates followed by a single *and* gate.

Key Results

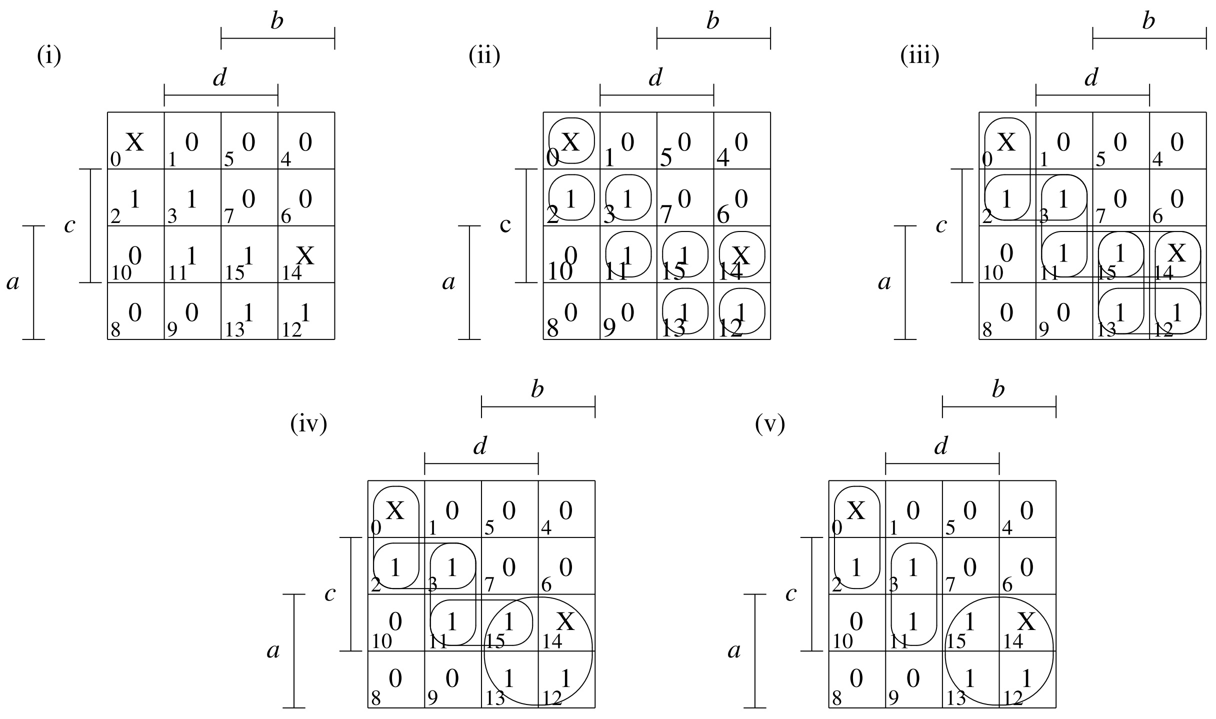
This survey will start by introducing the Karnaugh Map visualization technique, which will be used to assist in the subsequent explanation of the Quine–McCluskey algorithm for two-level Boolean minimization. This algorithm is optimal for its constrained problem variant. It is one of the fundamental algorithms in the field of computer-aided design and forms the basis or inspiration for many solutions to more general variants of the Boolean minimization problem.

Karnaugh Maps

Karnaugh Maps [4] provide a method of visualizing adjacency in Boolean space. A Karnaugh Map is a projection of an n-dimensional hypercube onto two-dimensional surface such that adjacent points in the hypercube remain adjacent in the two-dimensional projection. Figure 1 illustrates Karnaugh Maps of 1, 2, 3, and 4 variables: a, b, c, and d.



Two-Level Boolean Minimization, Figure 1
Boolean function spaces from one to four dimensions and their corresponding Karnaugh Maps



Two-Level Boolean Minimization, Figure 2
(i) Karnaugh Map of function *f*(*a*, *b*, *c*, *d*), (ii) elementary implicants, (iii) second-order implicants, (iv) prime implicants, and (v) a minimal cover

A *literal* is a single appearance of a complemented or uncomplemented input variable in a Boolean expression. A product term or *implicant* is the Boolean product, or *and*, of one or more literals. Every implicant corresponds to a repeated balanced bisection of Boolean space, or of the corresponding Karnaugh Map, i. e., an implicant is a rectangle in a Karnaugh Map with width m and height n where $m = 2^j$ and $n = 2^k$ for arbitrary non-negative integers j and k, e.g, the ovals in Fig. 2(ii–v). An *elementary implicant* is an implicant in which, for each variable of the corresponding function, the variable or its complement appears, e. g., the circles in Fig. 2(ii). Implicant *A covers* implicant *B* if every elementary implicant in *B* is also in *A*.

Prime implicants are implicants that are not covered by any other implicants, e. g., the ovals and circle in Fig. 2(iv). It is unnecessary to consider anything but prime implicants when seeking a minimal function representation because, if a non-prime implicants could be used to cover some set of elementary implicants, there is guaranteed to exist a prime implicant that covers those elementary implicants and contains fewer literals. One can draw the largest implicants covering each elementary implicant and covering no positions for which the function is *False*, thereby using Karnaugh Maps to identify prime implicants. One can then manually seek a compact subset of prime implicants covering all elementary implicants in the function.

This Karnaugh Map based approach is effective for functions with few inputs, i. e., those with low dimensionality. However, representing and manipulating Karnaugh Maps for functions of many variables is challenging. Moreover, the Karnaugh Map method provides no clear set of rules to follow when selecting a minimal subset of prime implicants to implement a function.

The Quine–McCluskey Algorithm

The Quine–McCluskey algorithm provides a formal, optimal way of solving the two-level Boolean minimization problem. W. V. Quine laid the essential theoretical groundwork for optimal two-level logic minimization [7,8]. However, E. J. McCluskey first proposed a precise algorithm to fully automate the process [6].

The Quine–McCluskey method has two phases: (1) produce all prime implicants and (2) select a minimal subset of prime implicants covering the function. In the first phase, the elementary implicants of a function are iteratively combined to produce implicants with fewer literals. Eventually, all prime implicants are thus produced. In the second phase, a minimal subset of prime implicants covering the on-set elementary implicants is selected using unate covering.

The Quine–McCluskey method may be illustrated using an example. Consider the function indicated by the Karnaugh Map in Fig. 2(i) and the truth table in Table 2. For each combination of Boolean input variable values, the function $f(a, b, c, d)$ is required to output a 0 (*False*), a 1 (*True*), or has no requirement. The lack of a requirement is indicated with an X, or don't-care symbol.

Expanding implicants as much as possible will ultimately produce the prime implicants. To do this, combine on-set and don't-care elementary implicants using the re-

Two-Level Boolean Minimization, Table 2
Truth table of function $f(a, b, c, d)$

Elementary implicant (a, b, c, d)	Function value (a, b, c, d)	Elementary implicant	Function value
0000	X	1000	0
0001	0	1001	0
0010	1	1010	0
0011	1	1011	1
0100	0	1100	1
0101	0	1101	1
0110	0	1110	X
0111	0	1111	1

Two-Level Boolean Minimization, Table 3
Identifying prime implicants

Number of ones	Elementary implicant (a, b, c, d)	Second-order implicant	Third-order implicant
0	0000 ✓	00X0	
1	0010 ✓	001X	
2	0011 ✓ 1100 ✓	X011 110X ✓ 11X0 ✓	11XX
3	1011 ✓ 1101 ✓ 1110 ✓	1X11 11X1 ✓ 111X ✓	
4	1111 ✓		

duction theorem ($\overline{a}b \vee ab = b$) shown in Table 1. The elementary implicants are circled in Fig. 2(ii) and listed in the second column of Table 3. In this table, 0s indicate complemented variables and 1s indicate uncomplemented variables, e. g., 0010 corresponds to $\overline{a}\overline{b}c\overline{d}$. It is necessary to determine all possible combinations of implicants. It is impossible to combine non-adjacent implicants, i. e., those that differ in more than one variable. Therefore, it is not necessary to consider combining any pair of implicants with a number of uncomplemented variables differing by any value other than 1. This fact can be exploited by organizing the implicants based on the number of ones they contain, as indicated by the first column in Table 3. All possible combinations of implicants in adjacent subsets are considered. For example, consider combining 0010 with 0011, which results in 001X or $\overline{a}\overline{b}c$, and also consider combining 0010 with 1100, which is impossible due to differences in more than one variable. Whenever an implicant is successfully merged, it is marked. These marked implicants are clearly not prime implicants because the implicants they produced cover them and contain fewer literals. Note that marked implicants should still be used for subsequent combinations. The merged implicants in

Two-Level Boolean Minimization, Table 4
Solving unate covering problem to select minimal cover

Requirements (elementary implicants)	Resources (prime implicants)				
	00X0	001X	X011	1X11	11XX
0010	✓	✓			
0011		✓	✓		
1011			✓	✓	
1100					✓
1101					✓
1111				✓	✓

the third column of Table 3 correspond to those depicted in Fig. 2(iii).

After all combinations of elementary implicants have been considered, and successful combinations listed in the third column, this process is repeated on the second-order merged implicants in the third column, producing the implicants in the fourth column. Implicants that contain don't-care marks in different locations may not be combined. This process is repeated until a column yielding no combinations is arrived at. The unmarked implicants in Table 3 are the prime implicants, which correspond to the implicants depicted in Fig. 2(iv).

After a function's prime implicants have been identified, it is necessary to select a minimal subset that covers the function. The problem can be formulated as unate covering. As shown in Table 4, label each column of a table with a prime implicant; these are resources that may be used to fulfill the requirements of the function. Label each row with an elementary implicant from the on-set; these rows correspond to requirements. Do not add rows for don't-cares. Don't-cares impose no requirements, although they were useful in simplifying prime implicants. Mark each row–column intersection for which the elementary implicant corresponding to the row is covered by the prime implicant corresponding to the column. If a column is selected, all the rows for which the column contains marks are *covered*, i. e., those requirements are satisfied. The goal is to cover all rows with a minimal-cost subset of columns. McCluskey defined minimal cost as having a minimal number of prime implicants, with ties broken by selecting the prime implicants containing the fewest literals. The most appropriate cost function depends on the implementation technology. One can also use a similar formulation with other cost functions, e. g., minimize the total number of literals by labeling each column with a cost corresponding to the number of literals in the corresponding prime implicant.

One can use a number of heuristics to accelerate solution of the unate covering problem, e. g., neglect rows that have a superset of the marks of any other row, for they will be implicitly covered and neglect columns that have a subset of the marks of any other column if their costs are as high, for the other column is at least as useful. One can easily select columns as long as there exists a row with only one mark because the marked column is required for a valid solution. However, there exist problem instances in which each row contains multiple marks. In the worst case, the best existing algorithms are required to make tentative decisions, determine the consequences, then backtrack and evaluate alternative decisions.

The unate covering problem appears in many applications. It is $\mathcal{NP}$-complete [5], even for the instances arising during two-level minimization [9]. Its use in the Quine–McCluskey method predates its categorization as an $\mathcal{NP}$-complete problem by 16 years. A detailed treatment of this problem would go well beyond the scope of this entry. However, Gimpel [3] as well as Coudert and Madre [2] provide good starting points for further reading.

Some families of logic functions have optimal two-level representations that grow in size exponentially in the number of inputs, but have more compact multi-level implementations. These families are frequently encountered in arithmetic, e. g., a function indicating whether the number of on inputs is odd. Efficient implementation of such functions requires manual design or multilevel minimization [1].

Applications

Digital computers are composed of precisely two things: (1) implementations of Boolean logic functions and (2) memory elements. The Quine–McCluskey method is used to permit efficient implementation of Boolean logic functions in a wide range of digital logic devices, including computers. The Quine–McCluskey method served as a starting point or inspiration for most currently-used logic minimization algorithms. Its direct use is contradicted when functions are not amenable to efficient two-level implementation, e. g., many arithmetic functions.

Cross References

► Local Approximation of Covering and Packing Problems

Recommended Reading

1. Brayton, R.K., Hachtel, G.D., Sangiovanni-Vincentelli, A.L.: Multi-level logic synthesis. Proc. IEEE **78**(2), 264–300 (1990)
2. Coudert, O., Madre, J.C.: New ideas for solving covering problems. In: Proc. Design Automation Conf., 1995, pp. 641–646

3. Gimpel, J.F.: A reduction technique for prime implicant tables. IEEE Trans. Electron. Comput. **14**(4), 535–541 (1965)
4. Karnaugh, M.: The map method for synthesis of combinational logic circuits. Trans. AIEE, Commun. Electron. **72**, 593–599 (1953)
5. Karp, R.M.: Reducibility among combinatorial problems. In: Miller, R.E., Thatcher, J.W. (eds.) Complexity of Computer Computations, pp. 85–103. Plenum Press, New York (1972)
6. McCluskey, E.J.: Minimization of Boolean functions. Bell Syst. Tech. J. **35**(6), 1417–1444 (1956)
7. Quine, W.V.: The problem of simplyfying truth functions. Am. Math. Mon. **59**(8), 521–531 (1952)
8. Quine, W.V.: A way to simplify truth functions. Am. Math. Mon. **62**(9), 627–631 (1955)
9. Umans, C., Villa, T., Sangiovanni-Vincentelli, A.L.: Complexity of two-level logic minimization. IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. **25**(7), 1230–1246 (2006)

Two-Person Game

▶ Complexity of Bimatrix Nash Equilibria

Two-Player Game

▶ Complexity of Bimatrix Nash Equilibria

Two-Player Nash

▶ Complexity of Bimatrix Nash Equilibria

U

Undirected Feedback Vertex Set

2005; Dehne, Fellows, Langston, Rosamond, Stevens
2005; Guo, Gramm, Hüffner, Niedermeier, Wernicke

JIONG GUO[1]
Department of Mathematics and Computer Science, University of Jena, Jena, Germany

Keywords and Synonyms

Odd cycle transversal

Problem Definition

The UNDIRECTED FEEDBACK VERTEX SET (UFVS) problem is defined as follows:

> **Input**: An undirected graph $G = (V, E)$ and an integer $k \geq 0$.
> **Task**: Find a *feedback vertex set* $F \subseteq V$ with $|F| \leq k$ such that each cycle in G contains at least one vertex from F. (The removal of all vertices in F from G results in a forest.)

Karp [11] showed that UFVS is NP-complete. Lund and Yannakakis [12] proved that there exists some constant $\epsilon > 0$ such that it is NP-hard to approximate the optimization version of UFVS to within a factor of $1 + \epsilon$. The best-known polynomial-time approximation algorithm for UFVS has a factor of 2 [1,4]. There is a simple and elegant randomized algorithm due to Becker et al. [3] which solves UFVS in $O(c \cdot 4^k \cdot kn)$ time on an n-vertex and m-edge graph by finding a feedback vertex set of size k with probability at least $1 - (1 - 4^{-k})^{c4^k}$ for an arbitrary constant c. An exact algorithm for UFVS with a running time of $O(1.7548^n)$ was recently found by Fomin et al. [9]. In the context of parameterized complexity [8,13], Bodlaender [5] and Downey and Fellows [7] were the first to show that the problem is *fixed-parameter tractable*, i. e., that the combinatorial explosion when solving it can be confined to the parameter k. The currently best fixed-parameter algorithm for UFVS runs in $O(c^k \cdot mn)$ for a constant c [6,10] (see [6] for the so far best running time analysis leading to a constant $c = 10.567$). This algorithm is the subject of this entry.

Key Results

The $O(c^k \cdot mn)$-time algorithm for the UNDIRECTED FEEDBACK VERTEX SET is based on the so-called "iterative compression" technique, which was introduced by Reed et al. [14]. The central observation of this technique is quite simple but fruitful: To derive a fixed-parameter algorithm for a minimization problem, it suffices to give a fixed-parameter "compression routine" that, given a size-$(k + 1)$ solution, either proves that there is no size-k solution or constructs one. Starting with a trivial instance and iteratively applying this compression routine a linear number of rounds to larger instances, one obtains a fixed-parameter algorithm of the problem. The main challenge of applying this technique to UFVS lies in showing that there is a fixed-parameter compression routine.

The compression routine from [6,10] works as follows:

1 Consider all possible partitions (X, Y) of the size-$(k + 1)$ feedback vertex set F with $|X| \leq k$ under the assumption that set X is entirely contained in the new size-k feedback vertex set F' and $Y \cap F' = \emptyset$
2 For each partition (X, Y), if the vertices in Y induce cycles, then answer "no" for this partition; otherwise, remove the vertices in X. Moreover, apply the following data reduction rules to the remaining graph:
- Remove degree-1 vertices.
- If there is a degree-2 vertex v with two neighbors v_1 and v_2, where $v_1 \notin Y$ or $v_2 \notin Y$, then remove v and connect v_1 and v_2. If this creates two parallel edges

[1]Supported by the Deutsche Forschungsgemeinschaft, Emmy Noether research group PIAF (fixed-parameter algorithms), NI 369/4

between v_1 and v_2, then remove the vertex of v_1 and v_2 that is not in Y and add it to any feedback vertex set for the reduced instance.

Finally, exhaustively examine every vertex set S with size at most $k - |X|$ of the reduced graph as to whether S can be added to X to form a feedback vertex set of the input graph. If there is one such vertex set, then output it together with X as the new size-k feedback vertex set.

The correctness of the compression routine follows from its brute-force nature and the easy to prove correctness of the two data reduction rules. The more involved part is to show that the compression routine runs in $O(c^k \cdot m)$ time: There are 2^k+1 partitions of F into the above sets (X, Y) and one can show that, for each partition, the reduced graph after performing the data reduction rules has at most $d \cdot k$ vertices for a constant d; otherwise, there is no size-k feedback vertex set for this partition. This then gives the $O(c^k \cdot m)$-running time. For more details on the proof of the $d \cdot k$-size bound see [6,10].

Given as input a graph G with vertex set $\{v_1, \ldots, v_n\}$, the fixed-parameter algorithm from [6,10] solves UFVS by iteratively considering the subgraphs $G_i := G[\{v_1, \ldots, v_i\}]$. For $i = 1$, the optimal feedback vertex set is empty. For $i > 1$, assume that an optimal feedback vertex set X_i for G_i is known. Obviously, $X_i \cup \{v_{i+1}\}$ is a solution set for G_{i+1}. Using the compression routine, the algorithm can in $O(c^k \cdot m)$ time either determine that $X_i \cup \{v_{i+1}\}$ is an optimal feedback vertex set for G_{i+1}, or, if not, compute an optimal feedback vertex set for G_{i+1}. For $i = n$, we thus have computed an optimal feedback vertex set for G in $O(c^k \cdot mn)$ time.

Theorem 1 UNDIRECTED FEEDBACK VERTEX SET *can be solved in $O(c^k \cdot mn)$ time for a constant c.*

Applications

The UNDIRECTED FEEDBACK VERTEX SET is of fundamental importance in combinatorial optimization. One typical application, for example, appears in the context of combinatorial circuit design [1]. For applications in the areas of constraint satisfaction problems and Bayesian inference, see Bar-Yehuda et al. [2].

Open Problems

It is open to explore the practical performance of the described algorithm. Another research direction is to improve the running time bound given in Theorem 1. Finally, it remains a long-standing open problem whether the FEEDBACK VERTEX SET on *directed* graphs is fixed-parameter tractable. The answer to this question would represent a significant breakthrough in the field.

Recommended Reading

1. Bafna, V., Berman, P., Fujito, T.: A 2-approximation algorithm for the undirected feedback vertex set problem. SIAM J. Discret. Math. **3**(2), 289–297 (1999)
2. Bar-Yehuda, R., Geiger, D., Naor, J., Roth, R.M.: Approximation algorithms for the feedback vertex set problem with applications to constraint satisfaction and Bayesian inference. SIAM J. Comput. **27**(4), 942–959 (1998)
3. Becker, A., Bar-Yehuda, R., Geiger, D.: Randomized algorithms for the Loop Cutset problem. J. Artif. Intell. Res. **12**, 219–234 (2000)
4. Becker, A., Geiger, D.: Approximation algorithms for the Loop Cutset problem. In: Proc. 10th Conference on Uncertainty in Artificial Intelligence, pp. 60–68. Morgan Kaufman, San Fransisco (1994)
5. Bodlaender, H.L.: On disjoint cycles. Int. J. Found. Comp. Sci. **5**(1), 59–68 (1994)
6. Dehne, F., Fellows, M.R., Langston, M.A., Rosamond, F., Stevens, K.: An $O(2^{O(k)} n^3)$ FPT algorithm for the undirected feedback vertex set problem. In: Proc. 11th COCOON. LNCS, vol. 3595, pp. 859–869. Springer, Berlin (2005). Long version to appear in: J. Discret. Algorithms
7. Downey, R.G., Fellows, M.R.: Fixed-parameter tractability and completeness. Congres. Numerant. **87**, 161–187 (1992)
8. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, Heidelberg (1999)
9. Fomin, F.V., Gaspers, S., Pyatkin, A.V.: Finding a minimum feedback vertex set in time $O(1.7548^n)$. In: Proc. 2th IWPEC. LNCS, vol. 4196, pp. 184–191. Springer, Berlin (2006)
10. Guo, J., Gramm, J., Hüffner, F., Niedermeier, R., Wernicke, S.: Compression-based fixed-parameter algorithms for Feedback Vertex Set and Edge Bipartization. J. Comp. Syst. Sci. **72**(8), 1386–1396 (2006)
11. Karp, R.: Reducibility among combinatorial problems. In: Miller, R., Thatcher, J. (eds.) Complexity of Computer Computations, pp. 85–103. Plenum Press, New York (1972)
12. Lund, C., Yannakakis, M.: The approximation of maximum subgraph problems. In: Proc. 20th ICALP. LNCS, vol. 700, pp. 40–51. Springer, Berlin (1993)
13. Niedermeier, R.: Invitation to Fixed-Parameter Algorithms. Oxford University Press, Oxford (2006)
14. Reed, B., Smith, K., Vetta, A.: Finding odd cycle transversals. Oper. Res. Lett. **32**(4), 299–301 (2004)

Unified Energy-Efficient Unicast and Broadcast Topology Control

► Degree-Bounded Planar Spanner with Low Weight

University Admissions Problem

► Hospitals/Residents Problem

Using Visualization in the Empirical Assessment of Algorithms

► Visualization Techniques for Algorithm Engineering

Utilitarian Mechanism Design for Single-Minded Agents

2005; Briest, Krysta, Vöcking

PIOTR KRYSTA[1], BERTHOLD VÖCKING[2]
[1] Department of Computer Science, University of Liverpool, Liverpool, UK
[2] Department of Computer Science, RWTH Aachen University, Aachen, Germany

Keywords and Synonyms

Forward (combinatorial, multi-unit) auction

Problem Definition

This problem deals with the design of efficiently computable incentive compatible, or truthful, mechanisms for combinatorial optimization problems with selfish one-parameter agents and a single seller. The focus is on approximation algorithms for NP-hard mechanism design problems. These algorithms need to satisfy certain monotonicity properties to ensure truthfulness.

A *one parameter agent* is an agent who as her private data has some *resource* as well as a *valuation*, i. e., the maximum amount of money she is willing to pay for this resource. Sometimes, however, the resource is assumed to be known to the mechanism. The scenario where a single seller offers these resources to the agents is primarily considered. Typically, the seller aims at maximizing the social welfare or her revenue. The work by Briest, Krysta and Vöcking [6] will mostly be considered, but also other existing models and results will be surveyed.

Utilitarian Mechanism Design

A famous example of mechanism design problems is given by combinatorial auctions (CAs), in which a single seller, auctioneer, wants to sell a collection of goods to potential buyers. A wider class of problems is encompassed by a *utilitarian mechanism design (maximization) problem* Π defined by a finite set of *objects* $\mathcal{A}$, a set of feasible outputs $O_\Pi \subseteq \mathcal{A}^n$ and a set of n agents. Each agent declares a set of objects $S_i \subseteq \mathcal{A}$ and a valuation function $v_i : \mathcal{P}(\mathcal{A}) \times \mathcal{A}^n \to \mathbb{R}$ by which she values all possible outputs. Given a vector $S = (S_1, \dots, S_n)$ of declarations one is interested in output $o^* \in O_\Pi$ maximizing the *social welfare*, i. e., $o^* \in \operatorname{argmax}_{o \in O_\Pi} \sum_{i=1}^n v_i(S_i, o)$. In CAs, an object a corresponds to a subset of goods. Each agent declares all the subsets she is interested in and the prices she would be willing to pay. An output specifies the sets to be allocated to the agents.

Here, a limited type of agents called *single-minded* is considered, introduced by Lehmann et al. [10]. Let $R_\preceq \subseteq \mathcal{A}^2$ be a reflexive and transitive relation on $\mathcal{A}$, such that there exists a special object $\varnothing \in \mathcal{A}$ with $\varnothing \preceq a$ for any $a \in \mathcal{A}$ to model the situation in which some agent does not contribute to the solution at all. For $a, b \in \mathcal{A}$ $(a, b) \in R_\preceq$ will be denoted by $a \preceq b$. The single-minded agent i declares a *single* object a_i and is fully defined by her *type* (a_i, v_i), with $a_i \in \mathcal{A}$ and $v_i > 0$. The valuation function introduced earlier reduces to

$$v_i(a_i, o) = \begin{cases} v_i\,, & \text{if } a_i \preceq o_i \\ 0\,, & \text{else}\,. \end{cases}$$

Agent i is called *known* if object a_i is known to the mechanism [11]. Here, mostly unknown agents will be considered. Intuitively, each a_i corresponds to an object agent i offers to contribute to the solution, v_i describes her valuation of any output o that indeed selects a_i. In CAs, relation $R_\preceq$ is set inclusion: an agent interested in set S will is also satisfied by S' with $S \subseteq S'$. For ease of notation let $(a, v) = ((a_1, v_1), \dots, (a_n, v_n))$, $(a_{-i}, v_{-i}) = ((a_1, v_1), \dots, (a_{i-1}, v_{i-1}), (a_{i+1}, v_{i+1}), \dots, (a_n, v_n))$ and $((a_i, v_i), (a_{-i}, v_{-i})) = (a, v)$.

Mechanism

A *mechanism* $M = (A, p)$ consists of an algorithm A computing a solution $A(a, v) \in O_\Pi$ and an n-tuple $p(a, v) = (p_1(a, v), \dots, p_n(a, v)) \in \mathbb{R}_+^n$ of payments collected from the agents. If $a_i \preceq A(a, v)_i$, agent i is *selected*, and let $S(A(a, v)) = \{i | a_i \preceq A(a, v)_i\}$ be the set of selected agents. Agent i's type is her private knowledge. Thus, the types declared by agents may not match their true types. To reflect this, let (a_i^*, v_i^*) refer to agent i's true type and (a_i, v_i) be the declared type. Given an output $o \in O_\Pi$, the *utility* of agent i is $u_i(a, v) = v_i(a_i^*, o) - p_i(a, v)$. Each agent's goal is to maximize her utility. To achieve this, she will try to manipulate the mechanism by declaring a false type if this could result in higher utility. A mechanism is called *truthful*, or *incentive compatible*, if no agent i can gain by lying about her type, i. e., given declarations (a_{-i}, v_{-i}), $u_i((a_i^*, v_i^*), (a_{-i}, v_{-i})) \geq u_i((a_i, v_i), (a_{-i}, v_{-i}))$ for any $(a_i, v_i) \neq (a_i^*, v_i^*)$.

```
Algorithm $A^k_\Pi$:
1  $\alpha_k := \frac{n}{\varepsilon \cdot 2^k}$;
2  for $i = 1, \ldots, n$ do
3      $v'_i := \min\{v_i, 2^{k+1}\}$;
4      $v''_i := \lfloor \alpha_k \cdot v'_i \rfloor$;
5  return $A_\Pi(a, v'')$;
```

```
Algorithm $A^{FPTAS}_\Pi$
1  $V := \max_i v_i$, $Best := (\emptyset, \ldots, \emptyset)$, $best := 0$;
2  for $j = 0, \ldots, \lceil \log(1-\varepsilon)^{-1} n \rceil + 1$ do
3      $k := \lfloor \log(V) \rfloor - j$;
4      if $w_k(A^k_\Pi(a, v)) > best$ then
5          $Best := A^k_\Pi(a, v)$; $best := w_k(A^k_\Pi(a, v))$;
6  return $Best$;
```

Utilitarian Mechanism Design for Single-Minded Agents, Figure 1
A monotone FPTAS for utilitarian problem Π and single-minded agents

Monotonicity

A sufficient condition for truthfulness of approximate mechanisms for single-minded CAs was first given by Lehmann et al. [10]. Their results can be adopted for the considered scenario. An algorithm A is *monotone* with respect to $R_\preceq$ if

$$\begin{aligned} & i \in S(A((a_i, v_i), (a_{-i}, v_{-i}))) \\ \Rightarrow\ & i \in S(A((a'_i, v'_i), (a_{-i}, v_{-i}))) \end{aligned}$$

for any $a'_i \preceq a_i$ and $v'_i \geq v_i$. Intuitively, one requires that a winning declaration (a_i, v_i) remains winning if an object a'_i, smaller according to $R_\preceq$, and a higher valuation v'_i are declared. If declarations (a_{-i}, v_{-i}) are fixed and object a_i declared by i, algorithm A defines a *critical value* θ^A_i, i. e., the minimum valuation v_i that makes (a_i, v_i) winning, i. e., $i \in S(A((a_i, v_i), (a_{-i}, v_{-i})))$ for any $v_i > \theta^A_i$ and $i \notin S(A((a_i, v_i), (a_{-i}, v_{-i})))$ for any $v_i < \theta^A_i$. The *critical value payment scheme* p^A associated with A is defined by $p^A_i(a, v) = \theta^A_i$, if $i \in S(A(a, v))$, and $p^A_i(a, v) = 0$, otherwise. The critical value for any fixed agent i can be computed, e. g., by performing binary search on interval $[0, v_i]$ and repeatedly running algorithm A to check if i is selected. Also, mechanism $M_A = (A, p^A)$ is *normalized*, i. e., agents that are not selected pay 0. Algorithm A is *exact*, if for declarations (a, v), $A(a, v)_i = a_i$ or $A(a, v)_i = \emptyset$ for all i. In analogy to [10] one obtains the following.

Theorem 1 *Let A be a monotone and exact algorithm for some utilitarian problem Π and single-minded agents. Then mechanism $M_A = (A, p^A)$ is truthful.*

Additional definitions

In the *unsplittable flow problem* (UFP), an undirected graph $G = (V, E)$, $|E| = m$, $|V| = n$, with edge capacities b_e, $e \in E$, and a set K of $k \geq 1$ commodities described by terminal pairs $(s_i, t_i) \in V \times V$ and a demand d_i and a value c_i are given. One assumes that $\max_i d_i \leq \min_e b_e$, $d_i \in [0, 1]$ for each $i \in K = \{1, \ldots, k\}$, and $b_e \geq 1$ for all $e \in E$. Let $B = \min_e\{b_e\}$. A feasible solution is a subset $K' \subseteq K$ and a single flow s_i-t_i-path for each $i \in K'$, such that the demands of K' can simultaneously and unsplittably be routed along the paths and the capacities are not exceeded. The goal in *UFP*, called *B-bounded UFP*, is to maximize the total value of the commodities in K'. A generalization is allocating bandwidth for multicast communication, where commodity is a set of terminals that should be connected by a multicast tree.

Key Results

Monotone approximation schemes

Let Π be a given utilitarian (maximization) problem. Given declarations (a, v), let $Opt(a, v)$ denote an optimal solution to Π on this instance and $w(Opt(a, v))$ the corresponding social welfare. Assuming that A_Π is a pseudopolynomial exact algorithm for Π an algorithm A^k_Π and monotone FPTAS for Π is defined in Fig. 1.

Theorem 2 *Let Π be a utilitarian mechanism design problem among single-minded agents, A_Π monotone pseudopolynomial algorithm for Π with running time $poly(n, V)$, where $V = \max_i v_i$, and assume that $V \leq w(Opt(a, v))$ for declaration (a, v). Then A^{FPTAS}_Π is a monotone FPTAS for Π.*

Theorem 2 can also be applied to minimization problems. Section "Applications" describes how these approximation schemes can be used for forward multi-unit auctions and job scheduling with deadlines.

Truthful primal-dual mechanisms

For an instance $G = (V, E)$ of UFP defined above, let S_i be the set of all s_i-t_i-paths in G, and $S = \bigcup_{i=1}^k S_i$. Given $S \in S_i$, let $q_S(e) = d_i$ if $e \in S$, and $q_S(e) = 0$ otherwise.

```
Algorithm Greedy-1:
1  $\mathcal{T} := \emptyset$; $K := \{1, \ldots, k\}$;
2  forall $e \in E$ do $y_e := 1/b_e$;
3  repeat
4      forall $i \in K$ do $S_i := \text{argmin}\left\{\sum_{e \in S} y_e \,\middle|\, S \in \mathcal{S}_i\right\}$;
5      $j := \text{argmax}\left\{\frac{c_i}{d_i \sum_{e \in S_i} y_e} \,\middle|\, i \in K\right\}$;
6      $\mathcal{T} := \mathcal{T} \cup \{S_j\}$; $K := K \setminus \{j\}$;
7      forall $e \in S_j$ do $y_e := y_e \cdot (\mathrm{e}^{B-1} m)^{q_{S_j}(e)/(b_e - 1)}$;
8  until $\sum_{e \in E} b_e y_e \geq \mathrm{e}^{B-1} m$ or $K = \emptyset$;
9  return $\mathcal{T}$.
```

Utilitarian Mechanism Design for Single-Minded Agents, Figure 2
Truthful mechanism for network (multicast) routing. e ≈ 2.718 is Euler number.

UFP is the following integer linear program (ILP)

$$\max \sum_{i=1}^{k} c_i \cdot \left(\sum_{S \in \mathcal{S}_i} x_S \right) \tag{1}$$

$$\text{s.t.} \quad \sum_{S : S \in \mathcal{S}, e \in S} q_S(e) x_S \leq b_e \quad \forall e \in E \tag{2}$$

$$\sum_{S \in \mathcal{S}_i} x_S \leq 1 \quad \forall i \in \{1, \ldots, k\} \tag{3}$$

$$x_S \in \{0, 1\} \quad \forall S \in \mathcal{S} . \tag{4}$$

The linear programming (LP) relaxation is the same linear program with constraints (4) replaced with $x_S \geq 0$ for all $S \in \mathcal{S}$. The corresponding dual linear program is

$$\min \sum_{e \in E} b_e y_e + \sum_{i=1}^{k} z_i \tag{5}$$

$$\text{s.t.} \quad z_i + \sum_{e \in S} q_S(e) y_e \geq c_i \quad \forall i \in \{1, \ldots, k\} \; \forall S \in \mathcal{S}_i \tag{6}$$

$$z_i, y_e \geq 0 \quad \forall i \in \{1, \ldots, k\} \; \forall e \in E. \tag{7}$$

Based on these LPs, Fig. 2 specifies a primal-dual mechanism for routing, called Greedy-1. Greedy-1 ensures feasibility by using y_e's: if an added set exceeded the capacity b_e of some $e \in E$, then this would imply the stopping condition already in the previous iteration. Using the weak duality of LPs the following result can be shown.

Theorem 3 *Greedy-1 outputs a feasible solution, and it is a* $(\frac{\mathrm{e}\gamma B}{B-1}(m)^{1/(B-1)})$*-approximation algorithm if there is a polynomial time algorithm that finds a* γ*-approximate set* S_i *in line 4.*

In case of UFP $\gamma = 1$, as the shortest s_i-t_i-path computation finds set S_i in line 4 of Greedy-1. For multicast routing, this problem corresponds to the NP-hard Steiner tree problem, for which one can take $\gamma = 1.55$. Greedy-1 can easily be shown to be monotone in demands and valuations as required in Theorem 1. Thus it implies a truthful mechanism for allocating network resources. The commodities correspond to bidders, the terminal nodes of bidders are known, but the bidders might lie about their demands and valuations. In the multicast routing the set of terminals for each bidder is known but the demands and valuations are unknown.

Corollary 1 *Given any* $\epsilon > 0$, $B \geq 1 + \epsilon$, *Greedy-1 is a truthful* $O(m^{1/(B-1)})$*-approximation mechanism for UFP (unicast routing) as well as for the multicast routing problem, where the demands and valuations of the bidders are unknown.*

When B is large, $\Omega(\log m)$, then the approximation factor in Corollary 1 becomes constant. Azar et al. [4] presented further results in case of large B. Awerbuch et al. [3] gave randomized online truthful mechanisms for uni- and multicast routing, obtaining an expected $O(\log(\mu m))$-approximation if $B = \Omega(\log m)$, where μ is the ratio of the largest to smallest valuation. Their approximation holds in fact with respect to the revenue of the auctioneer, but they assume that the demands are known to the mechanism. Bartal et al. [5] give a truthful $O(B \cdot (m/\theta)^{1/(B-2)})$-approximation mechanism for UFP with unknown valuations and demands, where $\theta = \min_i\{d_i\}$.

Greedy-1 can be modified to give truthful mechanisms for multi-unit CAs among unknown single-mined

```
Algorithm Greedy-2:
1   $\mathcal{T} := \emptyset$;
2   forall $e \in U$ do $y_e := 1/b_e$;
3   repeat
4       $S := \text{argmax}\left\{ \frac{c_S}{\sum_{e \in U} q_S(e) y_e} \;\middle|\; S \in \mathcal{S} \setminus \mathcal{T} \right\}$;
5       $\mathcal{T} := \mathcal{T} \cup \{S\}$;
6       forall $e \in S$ do $y_e := y_e \cdot (\mathrm{e}^B m)^{q_S(e)/b_e}$;
7   until $\sum_{e \in U} b_e y_e \geq \mathrm{e}^B m$;
8   return $\mathcal{T}$.
```

Utilitarian Mechanism Design for Single-Minded Agents, Figure 3
Truthful mechanism for multi-unit CAs among unknown single-minded bidders. For CAs without multisets: $q_S(e) \in \{0, 1\}$ for each $e \in U, S \in \mathcal{S}$.

bidders.[1] Archer et al. [2] used randomized rounding to obtain a truthful mechanism for multi-unit CAs, but only in a probabilistic sense and only for known bidders. Multi-unit CA among single-minded bidders is a special case of ILP (1)–(4), where $|\mathcal{S}_i| = 1$ for each $i \in K$, and $q_S(e) \in \{0, 1\}$ for each $e \in U$, $S \in \mathcal{S}$ (E is U in CAs). A bid of bidder $i \in K$ is $(a_i, v_i) = (S, c_S)$, $S \in \mathcal{S}_i$, and $c_S = c_i$ is the valuation. The relation $R_{\preceq}$ is $\subseteq$. Algorithm Greedy-2 in Fig. 3 is exact and monotone for CAs with unknown single-minded bidders, as needed in Theorem 1.

Theorem 4 *Algorithm Greedy-2 is a truthful $O(m^{\frac{1}{B}})$-approximation mechanism for multi-unit CAs among unknown single-minded bidders.*

Bartal et al. [5] presented a truthful mechanism for this problem among unknown single-minded bidders which is $O(B \cdot m^{1/(B-2)})$-approximate. (It works in fact for more general bidders.)

Applications

Applications of the techniques described above are presented and a short survey of other results.

Applications of monotone approximation schemes

In a *forward multi-unit auction* a single auctioneer wants to sell m identical items to n possible buyers (bidders). Each single-minded bidder specifies the number of items she is interested in and a price she is willing to pay. Elements in the introduced notation correspond to the requested and allocated numbers of items. Relation $R_{\preceq}$ describes that bidder i requesting q_i items will be satisfied also by any larger number of items. Mu'alem and Nisan [11] give a 2-approximate monotone algorithm for this problem. Theorem 2 gives a monotone FPTAS for multi-unit auctions among unknown single-minded bidders. This FPTAS is truthful with respect to agents where both the number of items and price are private.

In *job scheduling with deadlines (JSD)*, each agent i has a job with running time t_i, deadline d_i and a price v_i she is willing to pay if her job is processed by deadline d_i. Element a_i is defined as $a_i = (t_i, d_i)$. Output for agent i is a time slot for processing i's job. For two elements $a_i = (t_i, d_i)$ and $a'_i = (t'_i, d'_i)$ one has $a_i \preceq a'_i$ if $t_i \leq t'_i$ and $d_i \geq d'_i$. Theorem 2 leads to a monotone FPTAS, which, however, is not exact (see Theorem 1) with respect to deadlines, and so it is a truthful mechanism only if the deadlines are known. The techniques of Theorem 2 apply also to minimization mechanism design problems with a single buyer, such as reverse multi-unit auctions, scheduling to minimize tardiness, constrained shortest path and minimum spanning tree problems [6].

Applications of the primal dual algorithms

The applications of the primal dual algorithms are combinatorial auctions and auctions for unicast and multicast routing. As these applications are tied very much to the algorithms, they have already been presented in Sect. "Key Results".

Survey of other results

First truthful mechanisms for single-minded CAs were designed by Lehmann et al. [10], where they introduced the concept of single-minded agents, identified the role of monotonicity, and used greedy algorithms to design

[1]In the case of *unknown* single-minded bidders, the bidders have as private data not only their valuations (as in the case of *known* single-minded bidders) but also the sets they demand.

truthful mechanisms. Better approximation ratios of these greedy mechanisms were proved by Krysta [9] with the help of LP duality. A tool-box of techniques for designing truthful mechanisms for CAs was given by Mu'alem and Nisan [11].

The previous section presented a monotone FPTAS for job scheduling with deadlines where jobs are selfish agents and the seller offers the agents the facilities to process their jobs. Such scenarios when jobs are selfish agents to be scheduled on (possibly selfish) machines have been investigated further by Andelman and Mansour [1], see also references therein.

So far social welfare was mostly assumed as the objective, but for a seller probably more important is to maximize her revenue. This objective turns out to be much harder to enforce in mechanism design. Such truthful (in probabilistic sense) mechanisms were obtained for auctioning unlimited supply goods among one-parameter agents [7,8]. Another approach to maximizing seller's revenue is known as optimal auction design [12]. A seller wants to auction a single good among agents and each agent has a private value for winning the good. One assumes that the seller knows a joint distribution of those values and wants to maximize her expected revenue [13,14].

Cross References

Mechanisms that approximately maximize revenue for unlimited-supply goods as of Goldberg, Hartline and Wright 8 are presented in entry ▶ Competitive Auction.

Recommended Reading

1. Andelman, N., Mansour, Y.: A sufficient condition for truthfulness with single parameter agents. In: Proc. 8th ACM Conference on Electronic Commerce (EC),Ann, Arbor, Michigan, June (2006)
2. Archer, A., Papadimitriou, C.H., Talwar, K., Tardos, E.: An approximate truthful mechanism for combinatorial auctions with single parameter agents. In: Proc. 14th Ann. ACM–SIAM Symp. on Discrete Algorithms (SODA), pp. 205–214. Baltimore, Maryland (2003)
3. Awerbuch, B., Azar, Y., Meyerson, A.: Reducing truth-telling online mechanisms to online optimization. In: Proc. 35th Ann. ACM. Symp. on Theory of Comput. (STOC), San Diego, California (2003)
4. Azar, Y., Gamzu, I., Gutner, S.: Truthful unsplittable flow for large capacity networks. In: Proc. 19th Ann. ACM Symp. on Parallelism in Algorithms and Architectures (SPAA), pp. 320–329 (2007)
5. Bartal, Y., Gonen, R., Nisan, N.: Incentive compatible multi unit combinatorial auctions. In: Proceedings of the 9th conference on Theoretical aspects of rationality and knowledge (TARK), pp. 72–87. ACM Press (2003). http://doi.acm.org/10.1145/846241.846250
6. Briest, P., Krysta, P., Vöcking, B.: Approximation techniques for utilitarian mechanism design. In: Proc. 37th Ann. ACM. Symp. on Theory of Comput. (STOC), pp. 39–48 (2005)
7. Fiat, A., Goldberg, A.V., Hartline, J.D., Karlin, A.R.: Competitive generalized auctions. In: Proc. 34th Ann. ACM. Symp. on Theory of Comput. (STOC), pp. 72–81 (2002)
8. Goldberg, A.V., Hartline, J.D., Wright, A.: Competitive auctions and digital goods. In: Proc. 12th Ann. ACM–SIAM Symp. on Discrete Algorithms (SODA), pp. 735–744 (2001)
9. Krysta, P.: Greedy approximation via duality for packing, combinatorial auctions and routing. In: Proc. 30th Int. Conference on Mathematical Foundations of Comput. Sci. (MFCS). Lecture Notes in Computer Science, vol. 3618, pp. 615–627 (2005)
10. Lehmann, D.J., O'Callaghan, L.I., Shoham, Y.: Truth revelation in approximately efficient combinatorial auctions. In: Proc. 1st ACM Conference on Electronic Commerce (EC), pp. 96–102 (1999)
11. Mu'alem, A., Nisan, N.: Truthful approximation mechanisms for restricted combinatorial auctions. In: Proc. 18th Nat. Conf. Artificial Intelligence, pp. 379–384. AAAI (2002)
12. Myerson, R.B.: Optimal auction design. Math. Oper. Res. **6**, 58–73 (1981)
13. Ronen, A.: On approximating optimal auctions (extended abstract). In: Proc. 3rd ACM Conference on Electronic Commerce (EC), pp. 11–17 (2001)
14. Ronen, A., Saberi, A.: On the hardness of optimal auctions. In: Proc. 43rd Ann. IEEE Symp. on Foundations of Comput. Sci. (FOCS), pp. 396–405 (2002)

V

Valid-Utility Games

▶ Market Games and Content Distribution

VCG

▶ Generalized Vickrey Auction

Vector Sorting

▶ String Sorting

Vertex Coloring

▶ Distributed Vertex Coloring
▶ Exact Graph Coloring Using Inclusion–Exclusion

Vertex Cover Data Reduction

▶ Vertex Cover Kernelization

Vertex Cover Kernelization

2004; Abu-Khzam, Collins, Fellows, Langston, Suters, Symons

JIANER CHEN
Department of Computer Science,
Texas A&M University, College Station, TX, USA

Keywords and Synonyms

Vertex cover preprocessing; Vertex cover data reduction

Problem Definition

Let G be an undirected graph. A subset C of vertices in G is a *vertex cover* for G if every edge in G has at least one end in C. The (parametrized) VERTEX COVER problem is for each given instance (G, k), where G is a graph and $k \geq 0$ is an integer (the parameter), to determine whether the graph G has a vertex cover of at most k vertices.

The VERTEX COVER problem is one of the six "basic" NP-complete problems according to Garey and Johnson [4]. Therefore, the problem cannot be solved in polynomial time unless P = NP. However, the NP-completeness of the problem does not obviate the need for solving it because of its fundamental importance and wide applications. One approach was initiated based on the observation that in many applications, the parameter k is small. Therefore, by taking the advantages of this fact, one may be able to solve this NP-complete problem effectively and practically for instances with a small parameter. More specifically, algorithms of running time of the form $f(k)p(n)$ have been studied for VERTEX COVER, where $p(n)$ is a low-degree polynomial of the number $n = |G|$ of vertices in G and $f(k)$ is a function independent of n.

There has been an impressive sequence of improved algorithms for the VERTEX COVER problem. A number of new techniques have been developed during this research, including kernelization, folding, and refined branch-and-search. In particular, the *kernelization* method is the study of polynomial time algorithms that can significantly reduce the instance size for VERTEX COVER. The following are some concepts related to the kernelization method:

Definition 1 Two instances (G, k) and (G', k') of VERTEX COVER are *equivalent* if the graph G has a vertex cover of size $\leq k$ if and only if the graph G' has a vertex cover of size $\leq k'$.

Definition 2 A *kernelization algorithm* for the VERTEX COVER problem takes an instance (G, k) of VERTEX COVER as input and produces an equivalent instance (G', k') for the problem, such that $|G'| \leq |G|$ and $k' \leq k$.

The kernelization method has been used extensively in conjunction with other techniques in the development of algorithms for the VERTEX COVER problem. Two major issues in the study of kernelization method are (1) effective reductions of instance size; and (2) the efficiency of kernelization algorithms.

Key Results

A number of kernelization techniques are discussed and studied in the current paper.

Preprocessing Based on Vertex Degrees

Let (G, k) be an instance of VERTEX COVER. Let v be a vertex of degree larger than k in G. If a vertex cover C does not include v, then C must contain all neighbors of v, which implies that C contains more than k vertices. Therefore, in order to find a vertex cover of no more than k vertices, one must include v in the vertex cover, and recursively look for a vertex cover of $k - 1$ vertices in the remaining graph.

The following fact was observed on vertices of degree less than 3.

Theorem 1 *There is a linear time kernelization algorithm that on each instance (G, k) of* VERTEX COVER, *where the graph G contains a vertex of degree less than 3, produces an equivalent instance (G', k') such that $|G'| < |G|$ and/or $k < k'$.*

Therefore, vertices of high degree (i. e., degree $> k$) and low degree (i. e., degree < 3) can always be handled efficiently before any more time-consuming process.

Nemhauser-Trotter Theorem

Let G be a graph with vertices $v_1, v_2, \ldots, v_n$. Consider the following integer programming problem:

$$\begin{aligned} \text{(IP) Minimize} \quad & x_1 + x_2 + \cdots + x_n \\ \text{Subject to} \quad & x_i + x_j \geq 1 \quad \text{for each edge } [v_i, v_j] \text{ in } G \\ & x_i \in \{0, 1\}, \quad 1 \leq i \leq n \end{aligned}$$

It is easy to see that there is a one-to-one correspondence between the set of feasible solutions to (IP) and the set of vertex covers of the graph G. A natural LP-relaxation (LP) of the problem (IP) is to replace the restrictions $x_i \in \{0, 1\}$ with $x_i \geq 0$ for all i. Note that the resulting linear programming problem (LP) now can be solved in polynomial time.

Let $\sigma = \{x_1^0, \ldots, x_n^0\}$ be an optimal solution to the linear programming problem (LP). The vertices in the graph G can be partitioned into three disjoint parts according to σ:

$$\begin{aligned} I_0 &= \{v_i \mid x_i^0 < 0.5\}, \\ C_0 &= \{v_i \mid x_i^0 > 0.5\}, \text{ and} \\ V_0 &= \{v_i \mid x_i^0 = 0.5\} \end{aligned}$$

The following nice property of the above vertex partition of the graph G was first observed by Nemhauser and Trotter [5].

Theorem 2 ***(Nemhauser-Trotter)*** *Let $G[V_0]$ be the subgraph of G induced by the vertex set V_0. Then (1) every vertex cover of $G[V_0]$ contains at least $|V_0|/2$ vertices; and (2) every minimum vertex cover of $G[V_0]$ plus the vertex set C_0 makes a minimum vertex cover of the graph G.*

Let k be any integer, and let $G' = G[V_0]$ and $k' = k - |C_0|$. As first noted in [3], by Theorem 2, the instances (G, k) and (G', k') are equivalent, and $|G'| \leq 2k'$ is a necessary condition for the graph G' to have a vertex cover of size k'. This observation gives the following kernelization result.

Theorem 3 *There is a polynomial-time algorithm that for a given instance (G, k) for the* VERTEX COVER *problem, constructs an equivalent instance (G', k') such that $k' \leq k$ and $|G'| \leq 2k'$.*

A faster Nemhauser-Trotter Construction

Theorem 3 suggests a polynomial-time kernelization algorithm for VERTEX COVER. The algorithm is involved in solving the linear programming problem (LP) and partitioning the graph vertices into the sets I_0, C_0, and V_0. Solving the linear programming problem (LP) can be done in polynomial time but is kind of costly in particular when the input graph G is dense. Alternatively, Nemhauser and Trotter [5] suggested the following algorithm without using linear programming. Let G be the input graph with vertex set $\{v_1, \ldots, v_n\}$.

1. construct a bipartite graph B with vertex set $\{v_1^L, \ldots, v_n^L, v_1^R, \ldots, v_n^R\}$ such that $[v_i^L, v_j^R]$ is an edge in B if and only if $[v_i, v_j]$ is an edge in G;
2. find a minimum vertex cover C_B for B;
3. $I_0' = \{v_i \mid$ if neither v_i^L nor v_i^R is in $C_B\}$;
   $C_0' = \{v_i \mid$ if both v_i^L and v_i^R are in $C_B\}$;
   $V_0' = \{v_i \mid$ if exactly one of v_i^L and v_i^R is in $C_B\}$

It can be proved [5] (see also [2]) that Theorem 2 still holds true when the sets C_0 and V_0 in the theorem are replaced by the sets C_0' and V_0', respectively, constructed in the above algorithm.

The advantage of this approach is that the sets C'_0 and V'_0 can be constructed in time $O(m\sqrt{n})$ because the minimum vertex cover C_B for the bipartite graph B can be constructed via a maximum matching of B, which can be constructed in time $O(m\sqrt{n})$ using Dinic's maximum flow algorithm, which is in general faster than solving the linear programming problem (LP).

Crown Reduction

For a set S of vertices in a graph G, denote by $N(S)$ the set of vertices that are not in S but adjacent to some vertices in S. A *crown* in a graph G is a pair (I, H) of subsets of vertices in G satisfying the following conditions: (1) $I \neq \emptyset$ is an independent set, and $H = N(I)$; and (2) there is a matching M on the edges connecting I and H such that all vertices in H are matched in M.

It is quite easy to see that for a given crown (I, H), there is a minimum vertex cover that includes all vertices in H and excludes all vertices in I. Let G' be the graph obtained by removing all vertices in I and H from G. Then, the instances (G, k) and (G', k') are equivalent, where $k' = k - |H|$. Therefore, identification of crowns in a graph provides an effective way for kernelization.

Let G be the input graph. The following algorithm is proposed.

1. construct a maximal matching M_1 in G; let O be the set of vertices unmatched in M_1;
2. construct a maximum matching M_2 of the edges between O and $N(O)$; $i = 0$; let I_0 be the set of vertices in O that are unmatched in M_2;
3. repeat until $I_i = I_{i-1}$ $\{H_i = N(I_i)$; $I_{i+1} = I_i \cup N_{M_2}(H_i)$; $i = i + 1$; $\}$; (where $N_{M_2}(H_i)$ is the set of vertices in O that match the vertices in H_i in the matching M_2)
4. $I = I_i$; $H = N(I_i)$; output (I, H).

Theorem 4 *(1) if the set I_0 is not empty, then the above algorithm constructs a crown (I, H); (2) if both $|M_1|$ and $|M_2|$ are bounded by k, and $I_0 = \emptyset$, then the graph G has at most $3k$ vertices.*

According to Theorem 4, the above algorithm on an instance (G, k) of VERTEX COVER either (1) finds a matching of size larger than k – which implies that there is no vertex cover of k vertices in the graph G; or (2) constructs a crown (I, H) – which will reduce the size of the instance; or (3) in case neither of (1) and (2) holds true, concludes that the graph G contains at most $3k$ vertices. Therefore, repeatedly applying the algorithm either derives a direct solution to the given instance, or constructs an equivalent instance (G', k') with $k' \leq k$ and $|G'| \leq 3k'$.

Applications

The research of the current paper was directly motivated by authors' research in bioinformatics. It is shown that for many computational biological problems, such as the construction of phylogenetic trees, phenotype identification, and analysis of microarray data, preprocessing based on the kernelization techniques has been very effective.

Experimental Results

Experimental results are given for handling graphs obtained from the study of phylogenetic trees based on protein domains, and from the analysis of microarray data. The results show that in most cases the best way to kernelize is to start handling vertices of high and low degrees (i. e., vertices of degree larger than k or smaller than 3) before attempting any of the other kernelization techniques. Sometimes, kernelization based on Nemhauser-Trotter Theorem can solve the problem without any further branching. It is also observed that sometimes particularly on dense graphs, kernelization techniques based on Nemhauser-Trotter Theorem are kind of time-consuming but do not reduce the instance size by much. On the other hand, the techniques based on high-degree vertices and crown reduction seem to work better.

Data Sets

The experiments were performed on graphs obtained based on data from NCBI and SWISS-PROT, well known open-source repositories of biological data.

Cross References

► Data Reduction for Domination in Graphs
► Local Approximation of Covering and Packing Problems
► Vertex Cover Search Trees

Recommended Reading

1. Abu-Khzam, F., Collins, R., Fellows, M., Langston, M., Suters, W., Symons, C.: Kernelization algorithms for the vertex cover problem: theory and experiments. In: Proc. Workshop on Algorithm Engineering and Experiments (ALENEX) pp. 62–69 (2004)
2. Bar-Yehuda, R., Even, S.: A local-ratio theorem for approximating the weighted vertex cover problem. Ann. Discret. Math. **25**, 27–45 (1985)
3. Chen, J., Kanj, I.A., Jia, W.: Vertex cover: further observations and further improvements. J. Algorithm **41**, 280–301 (2001)
4. Garey, M., Johnson, D.: Computers and Intractability: A Guide to the Theory of NP-completeness. Freeman, San Francisco (1979)
5. Nemhauser, G.L., Trotter, L.E.: Vertex packing: structural properties and algorithms. Math. Program. **8**, 232–248 (1975)

Vertex Cover Preprocessing

► Vertex Cover Kernelization

Vertex Cover Search Trees

2001; Chen, Kanj, Jia

JIANER CHEN
Department of Computer Science, Texas A&M University, College Station, TX, USA

Keywords and Synonyms

Branch and search; Branch and bound

Problem Definition

The VERTEX COVER problem is one of the six "basic" NP-complete problems according to Garey and Johnson [7]. Therefore, the problem cannot be solved in polynomial time unless P = NP. However, the NP-completeness of the problem does not obviate the need for solving it because of its fundamental importance and wide applications.

One approach is to develop *parameterized algorithms* for the problem, with the computational complexity of the algorithms being measured in terms of both input size and a parameter value. This approach was initiated based on the observation that in many applications, the instances of the problem are associated with a small parameter. Therefore, by taking the advantages of the small parameters, one may be able to solve this NP-complete problem effectively and practically.

The problem is formally defined as follows. Let G be an (undirected) graph. A subset C of vertices in G is a *vertex cover* for G if every edge in G has at least one end in C. An instance of the (parameterized) VERTEX COVER problem consists of a pair (G, k), where G is a graph and k is an integer (the parameter), which is to determine whether the graph G has a vertex cover of k vertices. The goal is to develop parameterized algorithms of running time $O(f(k)p(n))$ for the VERTEX COVER problem, where $p(n)$ is a lower-degree polynomial of the input size n, and $f(k)$ is the non-polynomial part that is a function of the parameter k but independent of the input size n. It would be expected that the non-polynomial function $f(k)$ is as small as possible. Such an algorithm would become "practically effective" when the parameter value k is small. It should be pointed out that unless an unlikely consequence occurs in complexity theory, the function $f(k)$ is at least an exponential function of the parameter k [8].

Key Results

A number of techniques have been proposed in the development of parameterized algorithms for the VERTEX COVER problem.

Kernelization

Suppose (G, k) is an instance for the VERTEX COVER problem, where G is a graph and k is the parameter. The *kernelization* operation applies a polynomial time preprocessing on the instance (G, k) to construct another instance (G', k'), where G' is a smaller graph (the *kernel*) and $k' \leq k$, such that G' has a vertex cover of k' vertices if and only if G has a vertex cover of k vertices. Based on a classical result by Nemhauser and Trotter [9], the following kernelization result was derived.

Theorem 1 *There is an algorithm of running time $O(kn + k^3)$ that for a given instance (G, k) for the* VERTEX COVER *problem, constructs another instance (G', k') for the problem, where the graph G' contains at most $2k'$ vertices and $k' \leq k$, such that the graph G has a vertex cover of k vertices if and only if the graph G' has a vertex cover of k' vertices.*

Therefore, kernelization provides an efficient preprocessing for the VERTEX COVER problem, which allows one to concentrate on graphs of small size (i. e., graphs whose size is only related to k).

Folding

Suppose v is a degree-2 vertex in a graph G with two neighbors u and w such that u and w are not adjacent to each other. Construct a new graph G' as follows: remove the vertices v, u, and w and introduce a new vertex v_0 that is adjacent to all remaining neighbors of the vertices u and w in G. The graph G' is said being obtained from the graph G by *folding the vertex v*. The following result was derived.

Theorem 2 *Let G' be a graph obtained by folding a degree-2 vertex v in a graph G, where the two neighbors of v are not adjacent to each other. Then the graph G has a vertex cover of k vertices if and only if the graph G' has a vertex cover of $k - 1$ vertices.*

An folding operation allows one to decrease the value of the parameter k without branching. Therefore, folding operations are regarded as very efficient in the development of exponential time algorithms for the VERTEX COVER problem. Recently, the folding operation has be generalized to apply to a set of more than one vertex in a graph [6].

Branch and Search

A main technique is the *branch and search* method that has been extensively used in the development of algorithms for the VERTEX COVER problem (and for many other NP-hard problems). The method can be described as follows. Let (G, k) be an instance of the VERTEX COVER problem. Suppose that somehow a collection $\{C_1, \ldots, C_b\}$ of vertex subsets in the graph G is identified, where for each i, the subset C_i has c_i vertices, such that if the graph G contains a vertex cover of k vertices, then at least for one C_i of the vertex subsets in the collection, there is a vertex cover of k vertices for G that contains all vertices in C_i. Then a collection of (smaller) instances (G_i, k_i) can be constructed, where $1 \le i \le b$, $k_i = k - c_i$, and G_i is obtained from G by removing all vertices in C_i. Note that the original graph G has a vertex cover of k vertices if and only if for one (G_i, k_i) of the smaller instances the graph G_i has a vertex cover of k_i vertices. Therefore, now the process can be branched into b sub-processes, each on a smaller instance (G_i, k_i) recursively searches for a vertex cover of k_i vertices in the graph G_i.

Let $T(k)$ be the number of leaves in the search tree for the above branch and search process on the instance (G, k), then the above branch operation gives the following recurrence relation:

$$T(k) = T(k - c_1) + T(k - c_2) + \cdots + T(k - c_b)$$

To solve this recurrence relation, let $T(k) = x^k$ so that the above recurrence relation becomes

$$x^k = x^{k-c_1} + x^{k-c_2} + \cdots + x^{k-c_b}$$

It can be proved [3] that the above polynomial equation has a unique root x_0 larger than 1. From this, one gets $T(k) = x_0^k$, which, up to a polynomial factor, gives an upper bound on the running time of the branch and search process on the instance (G, k).

The simplest case is that a vertex v of degree $d > 0$ in the graph G is picked. Let $w_1, \ldots, w_d$ be the neighbors of v. Then either v is contained in a vertex cover C of k vertices, or, if v is not contained in C, then all neighbors $w_1, \ldots, w_d$ of v must be contained in C. Therefore, one obtains a collection of two subsets $C_1 = \{v\}$ and $C_2 = \{w_1, \ldots, w_d\}$, on which the branch and search process can be applied.

The efficiency of a branch and search operation depends on how effectively one can identify the collection of the vertex subsets. Intuitively, the larger the sizes of the vertex subsets, the more efficient is the operation. Much effort has been made in the development of VERTEX COVER algorithms to achieve larger vertex subsets. Improvements on the size of the vertex subsets have been involved with very complicated and tedious analysis and enumerations of combinatorial structures of graphs. The current paper [3] achieved a collection of two subsets C_1 and C_2 of sizes $c_1 = 1$ and $c_2 = 6$, respectively, and other collections of vertex subsets that are at least as good as this (the techniques of kernelization and vertex folding played important roles in achieving these collections). This gives the following algorithm for the VERTEX COVER problem.

Theorem 3 *The* VERTEX COVER *problem can be solved in time* $O(kn + 1.2852^k)$.

Very recently, a further improvement over Theorem 3 has been achieved that gives an algorithm of running time $O(kn + 1.2738^k)$ for the VERTEX COVER problem [4].

Applications

The study of parameterized algorithms for the VERTEX COVER problem was motivated by ETH Zürich's DARWIN project in computational biology and computational biochemistry (see, e. g. [10,11],). A number of computational problems in the project, such as multiple sequence alignments [10] and biological conflict resolving [11], can be formulated into the VERTEX COVER problem in which the parameter value is in general not larger than 100. Therefore, an algorithm of running time $O(kn + 1.2852^k)$ for the problem becomes very effective and practical in solving these problems.

The parameterized algorithm given in Theorem 3 has also induced a faster algorithm for another important NP-hard problem, the MAXIMUM INDEPENDENT SET problem on sparse graphs [3].

Open Problems

The main open problem in this line of research is how far one can go along this direction. More specifically, how small the constant $c > 1$ can be for the VERTEX COVER problem to have an algorithm of running time $O(c^k n^{O(1)})$? With further more careful analysis on graph combinatorial structures, it seems possible to slightly improve the current best upper bound [4] for the problem. Some new techniques developed more recently [6] also seem very promising to improve the upper bound. On the other hand, it is known that the constant c cannot be arbitrarily close to 1 unless certain unlikely consequence occurs in complexity theory [8].

Experimental Results

A number of research groups have implemented some of the ideas of the algorithm in Theorem 3 or its variations, including the Parallel Bioinformatics project in Carleton University [2], the High Performance Computing project in University of Tennessee [1], and the DARWIN project in ETH Zürich [10,11]. As reported in [5], these implementations showed that this algorithm and the related techniques are "quite practical" for the VERTEX COVER problem with parameter value k up to around 400.

Cross References

► Data Reduction for Domination in Graphs
► Local Approximation of Covering and Packing Problems
► Local Search Algorithms for kSAT
► Vertex Cover Kernelization

Recommended Reading

1. Abu-Khzam, F., Collins, R., Fellows, M., Langston, M., Suters, W., Symons, C.: Kernelization algorithms for the vertex cover problem: theory and experiments. Proc. Workshop on Algorithm Engineering and Experiments (NLENEX), pp. 62–69. (2004)
2. Cheetham, J., Dehne, F., Rau-Chaplin, A., Stege, U., Taillon, P.: Solving large FPT problems on coarse grained parallel machines. J. Comput. Syst. Sci. **67**, 691–706 (2003)
3. Chen, J., Kanj, I.A., Jia, W.: Vertex cover: further observations and further improvements. J. Algorithms **41**, 280–301 (2001)
4. Chen, J., Kanj, I.A., Xia, G.: Improved parameterized upper bounds for vertex cover. In: Lecture Notes in Computer Science (MFCS 2006), vol. 4162, pp. 238–249. Springer, Berlin (2006)
5. Fellows, M.: Parameterized complexity: the main ideas and some research frontiers. In: Lecture Notes in Computer Science (ISAAC 2001), vol. 2223, pp. 291–307. Springer, Berlin (2001)
6. Fomin, F., Grandoni, F., Kratsch, D.: Measure and conquer: a simple $O(2^{0.288n})$ independent set algorithm. In: Proc. 17th Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA 2006), pp. 18–25 (2006)
7. Garey, M., Johnson, D.: Computers and Intractability: A Guide to the Theory of NP-completeness. Freeman, San Francisco (1979)
8. Impagliazzo, R., Paturi, R.: Which problems have strongly exponential complexity? J. Comput. Syst. Sci. **63**, 512–530 (2001)
9. Nemhauser, G.L., Trotter, L.E.: Vertex packing: structural properties and algorithms. Math. Program. **8**, 232–248 (1975)
10. Roth-Korostensky, C.: Algorithms for building multiple sequence alignments and evolutionary trees. Ph. D. Thesis, ETH Zürich, Institute of Scientific Computing (2000)
11. Stege, U.: Resolving conflicts from problems in computational biology. Ph. D. Thesis, ETH Zürich, Institute of Scientific Computing (2000)

Vickrey–Clarke–Groves Mechanism

► Generalized Vickrey Auction

Visualization Techniques for Algorithm Engineering

2002; Demetrescu, Finocchi, Italiano, Näher

CAMIL DEMETRESCU, GIUSEPPE F. ITALIANO
Department of Computer & Systems Science,
University of Rome, Rome, Italy

Keywords and Synonyms

Using visualization in the empirical assessment of algorithms

Problem Definition

The whole process of designing, analyzing, implementing, tuning, debugging and experimentally evaluating algorithms can be referred to as *Algorithm Engineering*. Algorithm Engineering views algorithmics also as an engineering discipline rather than a purely mathematical discipline. Implementing algorithms and engineering algorithmic codes is a key step for the transfer of algorithmic technology, which often requires a high-level of expertise, to different and broader communities, and for its effective deployment in industry and real applications.

Experiments can help measure practical indicators, such as implementation constant factors, real-life bottlenecks, locality of references, cache effects and communication complexity, that may be extremely difficult to predict theoretically. Unfortunately, as in any empirical science, it may be sometimes difficult to draw general conclusions about algorithms from experiments. To this aim, some researchers have proposed accurate and comprehensive guidelines on different aspects of the empirical evaluation of algorithms maturated from their own experience in the field (see, for example [1,15,16,20]). The interested reader may find in [18] an annotated bibliography of experimental algorithmics sources addressing methodology, tools and techniques.

The process of implementing, debugging, testing, engineering and experimentally analyzing algorithmic codes is a complex and delicate task, fraught with many difficulties and pitfalls. In this context, traditional low-level textual debuggers or industrial-strength development environments can be of little help for algorithm engineers, who are mainly interested in high-level algorithmic ideas rather than in the language and platform-dependent details of actual implementations. Algorithm visualization environments provide tools for abstracting irrelevant program details and for conveying into still or animated images the high-level algorithmic behavior of a piece of software.

Among the tools useful in algorithm engineering, visualization systems exploit interactive graphics to enhance the development, presentation, and understanding of computer programs [27]. Thanks to the capability of conveying a large amount of information in a compact form that is easily perceivable by a human observer, visualization systems can help developers gain insight about algorithms, test implementation weaknesses, and tune suitable heuristics for improving the practical performances of algorithmic codes. Some examples of this kind of usage are described in [12].

Key Results

Systems for algorithm visualization have matured significantly since the rise of modern computer graphic interfaces and dozens of algorithm visualization systems have been developed in the last two decades [2,3,4,5,6,8,9,10,13,17,25,26,29]. For a comprehensive survey the interested reader can be referred to [11,27] and to the references therein. The remainder of this entry discusses the features of algorithm visualization systems that appear to be most appealing for their deployment in algorithm engineering.

Critical Issues

From the viewpoint of the algorithm developer, it is desirable to rely on systems that offer visualizations at a *high level of abstraction*. Namely, one would be more interested in visualizing the behavior of a complex data structure, such as a graph, than in obtaining a particular value of a given pointer.

Fast prototyping of visualizations is another fundamental issue: algorithm designers should be allowed to create visualization from the source code at hand with little effort and without heavy modifications. At this aim, *reusability* of visualization code could be of substantial help in speeding up the time required to produce a running animation.

One of the most important aspects of algorithm engineering is the development of *libraries*. It is thus quite natural to try to interface visualization tools to algorithmic software libraries: libraries should offer default visualizations of algorithms and data structures that can be refined and customized by developers for specific purposes.

Software visualization tools should be able to animate *not just "toy programs"*, but significantly complex algorithmic codes, and to test their behavior on large data sets. Unfortunately, even those systems well suited for large information spaces often lack advanced navigation techniques and methods to alleviate the screen bottleneck. Finding a solution to this kind of limitations is nowadays a challenge.

Advanced debuggers take little advantage of sophisticated graphical displays, even in commercial software development environments. Nevertheless, software visualization tools may be very beneficial in addressing problems such as finding memory leaks, understanding anomalous program behavior, and studying performance. In particular, environments that provide interpreted execution may more easily integrate advanced facilities in support to *debugging and performance monitoring*, and many recent systems attempt at exploring this research direction.

Techniques

One crucial aspect in visualizing the dynamic behavior of a running program is the way it is conveyed into graphic abstractions. There are two main approaches to bind visualizations to code: the event-driven and the state-mapping approach.

Event-Driven Visualization A natural approach to algorithm animation consists of annotating the algorithmic code with calls to visualization routines. The first step consists of identifying the relevant actions performed by the algorithm that are interesting for visualization purposes. Such relevant actions are usually referred to as *interesting events*. As an example, in a sorting algorithm the swap of two items can be considered an interesting event. The second step consists of associating each interesting event with a modification of a graphical scene. Animation scenes can be specified by setting up suitable visualization procedures that drive the graphic system according to the actual parameters generated by the particular event. Alternatively, these visualization procedures may simply log the events in a file for a *post-mortem* visualization. The calls to the visualization routines are usually obtained by annotating the original algorithmic code at the points where the interesting events take place. This can be done either by hand or by means of specialized editors. Examples of toolkits based on the event-driven approach are Polka [28] and *GeoWin*, a C++ data type that can be easily interfaced with algorithmic software libraries of great importance in algorithm engineering such as CGAL [14] and LEDA [19].

State Mapping Visualization Algorithm visualization systems based on state mapping rely on the assumption that observing how the variables change provides clues to the actions performed by the algorithm. The focus is on capturing and monitoring the data modifications rather than on processing the interesting events issued by the annotated algorithmic code. For this reason they are also referred to as "data driven" visualization systems. Conventional debuggers can be viewed as data driven systems, since they provide direct feedback of variable modifications. The main advantage of this approach over the event-driven technique is that a much greater ignorance of the code is allowed: indeed, only the interpretation of the variables has to be known to animate a program. On the other hand, focusing only on data modification may sometimes limit customization possibilities making it difficult to realize animations that would be natural to express with interesting events. Examples of tools based on the state mapping approach are Pavane [23,25], which marked the first paradigm shift in algorithm visualization since the introduction of interesting events, and Leonardo [10] an integrated environment for developing, visualizing, and executing C programs.

A comprehensive discussion of other techniques used in algorithm visualization appears in [7,21,22,24,27].

Applications

There are several applications of visualization in algorithm engineering, such as testing and debugging of algorithm implementations, visual inspection of complex data structures, identification of performance bottlenecks, and code optimization. Some examples of uses of visualization in algorithm engineering are described in [12].

Open Problems

There are many challenges that the area of algorithm visualization is currently facing. First of all, the real power of an algorithm visualization system should be in the hands of the final user, possibly inexperienced, rather than of a professional programmer or of the developer of the tool. For instance, instructors may greatly benefit from fast and easy methods for tailoring animations to their specific educational needs, while they might be discouraged from using systems that are difficult to install or heavily dependent on particular software/hardware platforms. In addition to being easy to use, a software visualization tool should be able to animate significantly complex algorithmic codes without requiring a lot of effort. This seems particularly important for future development of visual debuggers. Finally, visualizing the execution of algorithms on large data sets seems worthy of further investigation. Currently, even systems designed for large information spaces often lack advanced navigation techniques and methods to alleviate the screen bottleneck, such as changes of resolution and scale, selectivity, and elision of information.

Cross References

► Experimental Methods for Algorithm Analysis

Recommended Reading

1. Anderson, R.J.: The Role of Experiment in the Theory of Algorithms. In: Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 59, pp. 191–195. American Mathematical Society, Providence, RI (2002)
2. Baker, J., Cruz, I., Liotta, G., Tamassia, R.: Animating Geometric Algorithms over the Web. In: Proceedings of the 12th Annual ACM Symposium on Computational Geometry. Philadelphia, Pennsylvania, May 24–26, pp. C3–C4 (1996)
3. Baker, J., Cruz, I., Liotta, G., Tamassia, R.: The Mocha Algorithm Animation System. In: Proceedings of the 1996 ACM Workshop on Advanced Visual Interfaces. Gubbio, Italy, May 27–29, pp. 248–250 (1996)
4. Baker, J., Cruz, I., Liotta, G., Tamassia, R.: A New Model for Algorithm Animation over the WWW, ACM Comput. Surv. **27**, 568–572 (1996)
5. Baker, R., Boilen, M., Goodrich, M., Tamassia, R., Stibel, B.: Testers and Visualizers for Teaching Data Structures. In: Proceeding of the 13th SIGCSE Technical Symposium on Computer Science Education. New Orleans, March 24–28, pp. 261–265 (1999)
6. Brown, M.: Algorithm Animation. MIT Press, Cambridge, MA (1988)
7. Brown, M.: Perspectives on Algorithm Animation. In: Proceedings of the ACM SIGCHI'88 Conference on Human Factors in Computing Systems. Washington, D.C., May 15–19, pp. 33–38 (1988)
8. Brown, M.: Zeus: a System for Algorithm Animation and Multi-View Editing. In: Proceedings of the 7th IEEE Workshop on Visual Languages. Kobe, Japan, October 8–11, pp. 4–9 (1991)
9. Cattaneo, G., Ferraro, U., Italiano, G.F., Scarano, V.: Cooperative Algorithm and Data Types Animation over the Net.J.Visual Lang.Comp. 13(4): 391– (2002)
10. Crescenzi, P., Demetrescu, C., Finocchi. I., Petreschi, R.: Reversible Execution and Visualization of Programs with LEONARDO. J. Visual Lang. Comp. **11**, 125–150 (2000). Leonardo is available at: http://www.dis.uniroma1.it/~demetres/Leonardo/. Accessed 15 Jan 2008
11. Demetrescu, C.: Fully Dynamic Algorithms for Path Problems on Directed Graphs, Ph. D. thesis, Department of Computer and Systems Science, University of Rome "La Sapienza" (2001)
12. Demetrescu, C., Finocchi, I., Italiano, G.F., Näher, S.: Visualization in algorithm engineering: tools and techniques. In: Experimental Algorithm Design to Robust and Effizient Software. Lecture Notes in Computer Science, vol. 2547. Springer, Berlin, pp. 24–50 (2002)
13. Demetrescu, C., Finocchi, I., Liotta, G.: Visualizing Algorithms over the Web with the Publication-driven Approach. In: Proc. of the 4th Workshop on Algorithm Engineering (WAE'00), Saarbrücken, Germany, 5–8 September (2000)

14. Fabri, A., Giezeman, G., Kettner, L., Schirra, S., Schönherr, S.: The cgal kernel: A basis for geometric computation. In: Applied Computational Geometry: Towards Geometric Engineering Proceedings (WACG'96), Philadelphia. Philadelphia, PA, May 27–28, pp. 191–202 (1996)
15. Goldberg, A.: Selecting problems for algorithm evaluation. In: Proc. 3rd Workshop on Algorithm Engineering (WAE'99). LNCS, vol. 1668. London, United Kingdom, July 19–21, pp. 1–11 (1999)
16. Johnson, D.: A theoretician's guide to the experimental analysis of algorithms. In: Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 59, 215–250. American Mathematical Society, Providence, RI (2002)>
17. Malony, A., Reed, D.: Visualizing Parallel Computer System Performance. In: Simmons, M., Koskela, R., Bucher, I. (eds.) Instrumentation for Future Parallel Computing Systems. ACM Press, New York (1989) pp. 59–90
18. McGeoch, C.: A bibliography of algorithm experimentation. In: Data Struktures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 59, 251–254. American Mathematical Society, Providence, RI (2002)
19. Mehlhorn, K., Naher, S.: LEDA: A Platform of Combinatorial and Geometric Computing, ISBN 0–521-56329–1. Cambrige University Press, Cambrige (1999)
20. Moret, B.: Towards a discipline of experimental algorithmics. In: Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 59, 197–214. American Mathematical Society, Providence, RI (2002)
21. Myers, B.: Taxonomies of Visual Programming and Program Visualization. J. Visual Lang. Comp. **1**, 97–123 (1990)
22. Price, B., Baecker, R., Small, I.: A Principled Taxonomy of Software Visualization. J. Visual Lang. Comp. **4**, 211–266 (1993)
23. Roman, G., Cox, K.: A Declarative Approach to Visualizing Concurrent Computations. Computer **22**, 25–36 (1989)
24. Roman, G., Cox, K.: A Taxonomy of Program Visualization Systems. Computer **26**, 11–24 (1993)
25. Roman, G., Cox, K., Wilcox, C., Plun, J.: PAVANE: a System for Declarative Visualization of Concurrent Computations. J. Visual Lang. Comp. **3**, 161–193 (1992)
26. Stasko, J.: Animating Algorithms with X-TANGO. SIGACT News **23**, 67–71 (1992)
27. Stasko, J., Domingue, J., Brown, M., Price B.: Software Visualization: Programming as a Multimedia Experience. MIT Press, Cambridge, MA (1997)
28. Stasko, J., Kraemer, E.: A Methodology for Building Application-Specific Visualizations of Parallel Programs. J. Parall. Distrib. Comp. **18**, 258–264 (1993)
29. Tal, A., Dobkin, D.: Visualization of Geometric Algorithms. IEEE Trans. Visual. Comp. Graphics **1**, 194–204 (1995)

Voltage Scaling

► Speed Scaling

Voltage Scheduling

2005; Li, Yao

Minming Li
Department of Computer Science, City University of Hong Kong, Hong Kong, China

Keywords and Synonyms

Dynamic speed scaling

Problem Definition

This problem is concerned with scheduling jobs with as little energy as possible by adjusting the processor speed wisely. This problem is motivated by *dynamic voltage scaling (DVS)* (or *speed scaling*) technique, which enables a processor to operate at a range of voltages and frequencies. Since energy consumption is at least a quadratic function of the supply voltage (hence CPU frequency/speed), it saves energy to execute jobs as slowly as possible while still satisfying all timing constraints. The associated scheduling problem is referred to as min-energy DVS scheduling. Previous work showed that min-energy DVS schedule can be computed in cubic time. The work of Li and Yao [7] considers the discrete model where the processor can only choose its speed from a finite speed set. This work designs an $O(dn \log n)$ two-phase algorithm to compute the min-energy DVS schedule for the discrete model (d represents the number of speeds) and also proves a lower bound of $\Omega(n \log n)$ for the computation complexity.

Notations and Definitions

In variable voltage scheduling model, there are two important sets:

1. Set J (job set) consists of n jobs: $j_1, j_2, \dots j_n$. Each job j_k has three parameters as its information: a_k representing the arrival time of j_k, b_k representing the deadline of j_k and R_k representing the total CPU cycles required by j_k. The parameters satisfy $0 \le a_k < b_k \le 1$.
2. Set SD (speed set) consists of the possible speeds that can be used by the processor. According to the property of SD, the scheduling model is divided into the following two categories,
   Continuous Model: The set SD is the set of positive real numbers.
   Discrete Model: The set SD consists of d positive values: $s_1 > s_2 > \dots > s_d$.

A schedule S consists of the following two functions: $s(t)$ which specifies the processor speed at time t and $job(t)$

which specifies the job executed at time t. Both functions are piecewise constant with finitely many discontinuities.

A feasible schedule must give each job its required number of cycles between arrival time and deadline, therefore satisfying the property: $\int_{a_k}^{b_k} s(t)\delta(k, job(t))\,\mathrm{d}t = R_k$, where $\delta(i, j) = 1$ if $i = j$ and $\delta(i, j) = 0$ otherwise.

EDF principle defines an ordering on the jobs according to their deadlines. At any time t, among jobs j_k that are available for execution, that is, j_k satisfying $t \in [a_k, b_k)$ and j_k not yet finished by t, it is the job with minimum b_k that will be executed during $[t, t + \epsilon]$.

The power P, or energy consumed per unit of time, is a convex function of the processor speed. The energy consumption of a schedule $S = (s(t), job(t))$ is defined as $E(S) = \int_0^1 P(s(t))\,\mathrm{d}t$.

A schedule is called an optimal schedule if its energy consumption is the minimum possible among all the feasible schedules. Note that for the Continuous Model, optimal schedule uses the same speed for the same job.

The work of Li and Yao considers the problem of computing an optimal schedule for the Discrete Model under the following assumptions.

Assumptions

1. **Single Processor:** At any time t, only one job can be executed.
2. **Preemptive:** Any job can be interrupted during its execution.
3. **Non-Precedence:** There is no precedence relationship between any pair of jobs.
4. **Offline:** The processor knows the information of all the jobs at time 0.

This problem is called Min-Energy Discrete Dynamic Voltage Scaling (MEDDVS).

Problem 1 (MEDDVS$_{J,SD}$) INPUT: *Integer n, Set $J = \{j_1, j_2, \dots, j_n\}$ and $SD = \{s_1, s_2, \dots, s_d\}$. $j_k = \{a_k, b_k, R_k\}$.*

OUTPUT: *Feasible schedule $S = (s(t), job(t))$ that minimizes $E(S)$.*

Kwon and Kim [6] proved that the optimal schedule for the Discrete Model can be obtained by first calculating the optimal schedule for the Continuous Model and then individually adjusting the speed of each job appropriately to adjacent levels in set SD. The time complexity is $O(n^3)$.

Key Results

The work of Li and Yao finds a direct approach for solving the MEDDVS problem without first computing the optimal schedule for the continuous model.

Definition 1 An s-schedule for J is a schedule which conforms to the EDF principle and uses constant speed s in executing any job of J.

Lemma 1 *The s-schedule for J can be computed in $O(n \log n)$ time.*

Definition 2 Given a job set J and any speed s, let $J^{\geq s}$ and $J^{<s}$ denote the subset of J consisting of jobs whose executing speeds are $\geq s$ and $< s$, respectively, in the optimal schedule for J in the Continuous Model. The partition $\langle J^{\geq s}, J^{<s}\rangle$ is referred to as the s-partition of J.

By extracting information from the s-schedule, a partition algorithm is designed to prove the following lemma:

Lemma 2 *The s-partition of J can be computed in $O(n \log n)$ time.*

By applying s-partition to J using all the d speeds in SD consecutively, one can obtain d subsets $J_1, J_2, \dots, J_d$ of J where jobs in the same subset J_i use the same two speeds s_i and s_{i+1} in the optimal schedule for the Discrete Model ($s_{d+1} = 0$).

Lemma 3 *Optimal schedule for job set J_i using speeds s_i and s_{i+1} can be computed in $O(n \log n)$ time.*

Combining the above three lemmas together, the main theorem follows:

Theorem 4 *The min-energy discrete DVS schedule can be computed in $O(dn \log n)$ time.*

A lower bound to compute the optimal schedule for the *Discrete Model* under the algebraic decision tree model is also shown by Li and Yao.

Theorem 5 *Any deterministic algorithm for computing min-energy discrete DVS schedule with $d \geq 2$ voltage levels requires $\Omega(n \log n)$ time for n jobs.*

Applications

Currently, dynamic voltage scaling technique is being used by the world's largest chip companies, e. g., Intel's SpeedStep technology and AMD's PowerNow technology. Although the scheduling algorithms being used are mostly online algorithms, offline algorithms can still find their places in real applications. Furthermore, the techniques developed in the work of Li and Yao for the computation of optimal schedules may have potential applications in other areas.

People also study energy efficient scheduling problems for other kind of job sets. Yun and Kim [10] proved that it is NP-hard to compute the optimal schedule for jobs with priorities and gave an FPTAS for that problem. Aydin et al. [1] considered energy efficient scheduling for real time periodic jobs and gave an $O(n^2 \log n)$ scheduling algorithm. Chen et al. [4] studied the weakly discrete model for non-preemptive jobs where speed is not allowed to change during the execution of one job. They proved the NP-hardness to compute the optimal schedule.

Another important application for this work is to help investigating scheduling model with more hardware restrictions (Burd and Brodersen [3] explained various design issues that may happen in dynamic voltage scaling). Besides the single processor model, people are also interested in the multiprocessor model [11].

Open Problems

A number of problems related to the work of Li and Yao remain open. In the Discrete Model, Li and Yao's algorithm for computing the optimal schedule requires time $O(dn \log n)$. There is a gap between this and the currently known lower bound $\Omega(n \log n)$. Closing this gap when considering d as an variable is an open problem.

Another open research area is the computation of the optimal schedule for the Continuous Model. Li, Yao and Yao [8] obtained an $O(n^2 \log n)$ algorithm for computing the optimal schedule. The bottleneck for the $\log n$ factor is in the computation of s-schedules. Reducing the time complexity for computing s-schedules is an open problem. It is also possible to look for other methods to deal with the Continuous Model.

Cross References

▶ List Scheduling
▶ Load Balancing
▶ Parallel Algorithms for Two Processors Precedence Constraint Scheduling
▶ Shortest Elapsed Time First Scheduling

Recommended Reading

1. Aydin, H., Melhem, R., Mosse, D., Alvarez, P.M.: Determining Optimal Processor Speeds for Periodic Real-Time Tasks with Different Power Characteristics. Euromicro Conference on Real-Time Systems, pp. 225–232. IEEE Computer Society, Washington, DC, USA (2001)
2. Bansalm, N., Kimbrel, T., Pruhs, K.: Dynamic Speed Scaling to Manage Energy and Temperature, Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science, pp. 520–529. IEEE Computer Society, Washington, DC, USA (2004)
3. Burd, T.D., Brodersen, R.W.: Design Issues for Dynamic Voltage Scaling, Proceedings of the 2000 international symposium on Low power electronics and design, pp. 9–14. ACM, New York, USA (2000)
4. Chen, J.J., Kuo, T.W., Lu, H.I.: Power-Saving Scheduling for Weakly Dynamic Voltage Scaling Devices Workshop on Algorithms and Data Structures (WADS). LNCS, vol. 3608, pp. 338–349. Springer, Berlin, Germany (2005)
5. Irani, S., Pruhs, K.: Algorithmic Problems in Power Management. ACM SIGACT News **36**(2), 63–76. New York, NY, USA (2005)
6. Kwon, W., Kim, T.: Optimal Voltage Allocation Techniques for Dynamically Variable Voltage Processors. ACM Trans. Embed. Comput. Syst. **4**(1), 211–230. New York, NY, USA (2005)
7. Li, M., Yao, F.F.: An Efficient Algorithm for Computing Optimal Discrete Voltage Schedules. SIAM J. Comput. **35**(3), 658–671. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA (2005)
8. Li, M., Yao, A.C., Yao, F.F.: Discrete and Continuous Min-Energy Schedules for Variable Voltage Processors, Proceedings of the National Academy of Sciences USA, 103, pp. 3983–3987. National Academy of Science of the United States of America, Washington, DC, USA (2005)
9. Yao, F., Demers, A., Shenker, S.: A Scheduling Model for Reduced CPU Energy, Proceedings of the 36th Annual IEEE Symposium on Foundations of Computer Science, pp. 374–382. IEEE Computer Society, Washington, DC, USA (1995)
10. Yun, H.S., Kim, J.: On Energy-Optimal Voltage Scheduling for Fixed-Priority Hard Real-Time Systems. ACM Trans. Embed. Comput. Syst. **2**, 393–430. ACM, New York, NY, USA (2003)
11. Zhu, D., Melhem, R., Childers, B.: Scheduling with Dynamic Voltage/Speed Adjustment Using Slack Reclamation in Multi-Processor RealTime Systems. Proceedings of the 22nd IEEE Real-Time Systems Symposium (RTSS'01), pp. 84–94. IEEE Computer Society, Washington, DC, USA (2001)

Voting Systems

▶ Quorums

W

Wait-Free Consensus

► Asynchronous Consensus Impossibility

Wait-Free Registers

► Registers

Wait-Free Renaming

► Renaming
► Topology Approach in Distributed Computing

Wait-Free Shared Variables

► Registers

Wait-Free Synchronization

1991; Herlihy

Mark Moir
Sun Microsystems Laboratories, Burlington, MA, USA

Problem Definition

The traditional use of locking to maintain consistency of shared data in concurrent programs has a number of disadvantages related to software engineering, robustness, performance, and scalability. As a result, a great deal of research effort has gone into *nonblocking* synchronization mechanisms over the last few decades.

Herlihy's seminal paper *Wait-Free Synchronization* [12] studied the problem of implementing concurrent data structures in a *wait-free manner*, i. e., so that every operation on the data structure completes in a finite number of steps by the invoking thread, regardless of how fast or slow other threads run and even if some or all of them halt permanently. Implementations based on locks are not wait-free because, while one thread holds a lock, others can take an unbounded number of steps waiting to acquire the lock. Thus, by requiring implementations to be wait-free, some of the disadvantages of locks may potentially be eliminated.

The first part of Herlihy's paper examined the power of different synchronization primitives for wait-free computation. He defined the *consensus number* of a given primitive as the maximum number of threads for which we can solve wait-free consensus using that primitive (together with read-write registers). The consensus problem requires participating threads to *agree* on a value (e. g., true or false) amongst values proposed by the threads. The ability to solve this problem is a key indicator of the power of synchronization primitives because it is central to many natural problems in concurrent computing. For example, in a software transactional memory system, threads must agree that a particular transaction either committed or aborted.

Herlihy established a *hierarchy* of synchronization primitives according to their consensus number. He showed (i) that the consensus number of read-write registers is 1 (so wait-free consensus cannot be solved for even two threads), (ii) that the consensus number of stacks and FIFO queues is 2, and (iii) that there are so-called *universal* primitives, which have consensus number ∞. Common examples include `compare-and-swap` (CAS) and the `load-linked/store-conditional` (LL/SC) pair.

There are a number of papers which examine Herlihy's hierarchy in more detail. These show that seemingly minor variations in the model or in the semantics of primitives can have a surprising effect on results. Most of this work is primarily of theoretical interest. The key practical point to take away from Herlihy's hierarchy is that we need universal primitives to support effective wait-free synchronization in general. Recognizing this fact, all modern shared-memory multiprocessors provide some form of universal primitive.

Herlihy additionally showed that a solution to consensus can be used to implement any shared object in a wait-free manner, and thus that any universal primitive suffices for this purpose. He demonstrated this idea using a so-called *universal construction*, which takes sequential code for an object and creates a wait-free implementation of the object using consensus to resolve races between concurrent operations. Despite the important practical ramifications of this result, the universal construction itself was quite impractical. The basic idea was to build a list of operations, using consensus to determine the order of operations, and to allow threads to iterate over the list applying the operations in order to determine the current state of the object. The construction required $O(N^3)$ space to ensure enough operations are retained to allow the current state to be determined. It was also very slow, requiring many threads to recompute the same information, and thus preventing parallelism between operations in addition.

Later, Herlihy [13] presented a more concrete universal construction based on the `LL`/`SC` instruction pair. This construction required $N + 1$ copies of the object for N threads and still did not admit any parallelism; thus it was also not practical. Despite this, work following on from Herlihy's has brought us to the point today that we can support practical programming models that provide nonblocking implementations of arbitrary shared objects. The remainder of this chapter discusses the state of nonblocking synchronization today, and mentions some history along the way.

Weaker Nonblocking Progress Conditions

Various researchers, including us, have had some success attempting to overcome the disadvantages of Herlihy's wait-free constructions. However, the results remain impractical due to excessive overhead and overly complicated algorithms. In fact, there are still no nontrivial wait-free shared objects in widespread practical use, either implemented directly or using universal constructions.

The biggest advances towards practicality have come from considering weaker progress conditions. While theoreticians worked on wait-free implementations, more pragmatic researchers sought lock-free implementations of shared objects. A *lock-free* implementation guarantees that, after a finite number of steps of any operation, *some* operation completes. In contrast to wait-free algorithms, it is in principle possible for one operation of a lock-free data structure to be continually starved by others. However, this rarely occurs in practice, especially because contention control techniques such as exponential backoff [1] are often used to reduce contention when it occurs, which makes repeated interference even more unlikely. Thus, the lack of a strong progress guarantee like wait-freedom has often been found to be acceptable in practice.

The observation that weaker nonblocking progress conditions allow simpler and more practical algorithms led Herlihy et al. [15] to define an even weaker condition: An *obstruction-free* algorithm does not guarantee that an operation completes unless it eventually encounters no more interference from other operations. In our experience, obstruction-free algorithms are easier to design, simpler, and faster in the common uncontended case than lock-free algorithms. The price paid for these benefits is that obstruction-free algorithms can "livelock", with two or more operations repeatedly interfering with each other forever. This is not merely a theoretical concern: it has been observed to occur in practice [16]. Fortunately, it is usually straightforward to eliminate livelock in practice through contention control mechanisms that control and manipulate when operations are executed to avoid repeated interference.

The obstruction-free approach to synchronization is thus to design simple and efficient algorithms for the weak obstruction-free progress condition, and to integrate orthogonal contention control mechanisms to facilitate progress when necessary. By largely separating the difficult issues of correctness and progress, we significantly ease the task of designing effective nonblocking implementations: the algorithms are not complicated by tightly coupled mechanisms for achieving lock-freedom, and it is easy to modify and experiment with contention control mechanisms because they are separate from the algorithm and do not affect its correctness. We have found this approach to be very powerful.

Transactional Memory

The severe difficulty of designing and verifying correct nonblocking data structures has led researchers to investigate the use of tools to produce them, rather than designing them directly. In particular, *transactional memory* ([5,17,23]) has emerged as a promising direction. Transactional memory allows programmers to express sections of code that should be executed atomically, and the transactional memory system (implemented in hardware, software, or a combination of the two) is responsible for managing interactions between concurrent transactions to ensure this atomicity. Here we concentrate on software transactional memory (STM).

The progress guarantee made by a concurrent data structure implemented using STM depends on the STM implementation. It is possible to characterize the progress conditions of transactional memory implementations in terms of a system of threads in which each operation on a shared data structure is executed by repeatedly attempting to apply it using a transaction until an attempt successfully commits. In this context, say the transactional memory implementation is obstruction-free if it guarantees that, if a thread repeatedly executes transactions and eventually encounters no more interference from other threads, then it eventually successfully commits a transaction.

Key Results

This section briefly discusses some of the most relevant results concerning nonblocking synchronization, and obstruction-free synchronization in particular.

While progress towards practicality was made with lock-free implementations of shared objects as well as lock-free STM systems, this progress was slow because simultaneously ensuring correctness and lock-freedom proved difficult. Before the introduction of obstruction-freedom, the lock-free STMs still had some severe disadvantages such as the need to declare and initialize all memory to be accessed by transactions in advance, the need for transactions to know in advance which memory locations they will access, unacceptable constraints on the layout of such memory, etc.

In addition to the work on tools such as STM for building nonblocking data structures, there has been a considerable amount of work on direct implementations. While this work has not yielded any practical wait-free algorithms, a handful of practical lock-free implementations for simple data structures such as queues and stacks have been achieved [21,24]. There are also a few slightly more ambitious implementations in the literature that are arguably practical, but the algorithms are complicated and subtle, many are incorrect, and almost none has a formal proof. Proofs for such algorithms are challenging, and minor changes to the algorithm require the proofs to be redone.

The next section, discusses some of the results that have been achieved by applying the obstruction-free approach. The remainder of this section, briefly discusses a few results related to the approach itself.

An important practical aspect of using an obstruction-free algorithm is how contention is managed when it arises. In introducing obstruction-freedom, Herlihy et al. [15] explained that contention control is necessary to facilitate progress in the face of contention because obstruction-free algorithms do not directly make any progress guarantee in this case. However, they did not directly address *how* contention control mechanisms could be used in practice.

Subsequently, Herlihy et al. [16] presented a dynamic STM system (see next section) that provides an interface for a modular contention manager, allowing for experimentation with alternative contention managers. Scherer and Scott [22] experimented with a number of alternatives, and found that the best contention manager depends on the workload. Guerraoui et al. [9] described an implementation that supports changing contention managers on the fly in response to changing workload conditions.

All of the contention managers discussed in the above-mentioned papers are ad hoc contention managers based on intuition; no analysis is given of what guarantees (if any) are made by the contention managers. Guerraoui et al. [10] made a first step towards a formal analysis of contention managers by showing that their `Greedy` contention manager guarantees that every transaction eventually completes. However, using the `Greedy` contention manager results in a blocking algorithm, so their proof necessarily assumes that threads do not fail while executing transactions.

Fich et al. [7] showed that any obstruction-free algorithm can be automatically transformed into one that is *practically* wait-free in any real system. "Practically" is said because the wait-free progress guarantee depends on partial synchrony that exists in any real system, but the transformed algorithm is not technically wait-free, because this term is defined in the context of a fully asynchronous system. Nonetheless, an algorithm achieved by applying the transformation of Fich et al. to an obstruction-free algorithm does guarantee progress to non-failed transactions, even if other transactions fail.

Work on incorporating contention management techniques into obstruction-free algorithms has mostly been done in the context of STM, so the contention manager can be called directly from the STM implementation. Thus, the programmer using the STM need not be concerned with how contention management is integrated, but this does not address how contention management is integrated into direct implementations of obstruction-free data structures.

One option is for the programmer to manually insert calls to a contention manager, but this approach is tedious and error prone. Guerraoui et al. [11] suggested a version of this approach in which the contention manager is abstracted out as a failure detector. They also explored what progress guarantees can be made by what failure detectors.

Attiya et al. [4] and Aguilera et al. [2] suggested changing the semantics of the data structure's operations so that they can return a special value in case of contention, thus allowing contention management to be done outside the data structure implementation. These approaches still leave a burden on the programmer to ensure that these special values are always returned by an operation that cannot complete due to contention, and that the correct special value is returned according to the prescribed semantics.

Another option is to use system support to ensure that contention management calls are made frequently enough to ensure progress. This support could be in the form of compiled-in calls, runtime support, signals sent upon expiration of a timer, etc. But all of these approaches have disadvantages such as not being applicable in general purpose environments, not being portable, etc.

Given that it remains challenging to design and verify direct obstruction-free implementations of shared data structures, and that there are disadvantages to the various proposals for integrating contention control mechanisms into them, using tools such as STMs with built-in contention management interfaces is the most convenient way to build nonblocking data structures.

Applications

The obstruction-free approach to nonblocking synchronization was introduced by Herlihy et al. [15], who used it to design a double-ended queue (deque) based on the widely available `CAS` instruction. All previous nonblocking deques either require exotic synchronization instructions such as `double-compare-and-swap` (`DCAS`), or have the disadvantage that operations at opposite ends of the queue always interfere with each other.

Herlihy et al. [16] introduced Dynamic STM (DSTM), the first STM that is dynamic in the following two senses: new objects can be allocated on the fly and subsequently accessed by transactions, and transactions do not need to know in advance what objects will be accessed. These two advantages made DSTM much more useful than previous STMs for programming dynamic data structures. As a result, nonblocking implementations of sophisticated shared data structures such as balanced search trees, skip lists, dynamic hash tables, etc. were suddenly possible.

The obstruction-free approach played a key role in the development of both of the results mentioned above: Herlihy et al. [16] could concentrate on the functionality and correctness of DSTM without worrying about how to achieve stronger progress guarantees such as lock-freedom.

The introduction of DSTM and of the obstruction-free approach have led to numerous improvements and variations by a number of research groups, and most of these have similarly followed the obstruction-free approach. However, Harris and Fraser [8] presented a dynamic STM called OSTM with similar advantages to DSTM, but it is lock-free. Experiments conducted at the University of Rochester [20] showed that DSTM outperformed OSTM by an order of magnitude on some workloads, but that OSTM outperformed DSTM by a factor of 2 on others. These differences are probably due to various design decisions that are (mostly) orthogonal to the progress condition, so it is not clear what we can conclude about how the choice of progress condition affects performance in this case.

Perhaps a more direct comparison can be made between another pair of algorithms, again an obstruction-free one by Herlihy et al. [14] and a similar but lock-free one by Harris and Fraser [8]. These algorithms, invented independently of each other, implement `MCAS` (`CAS` generalized to access M independently chosen memory locations). The two algorithms are very similar, and a close comparison revealed that the only real differences between them were due to Harris and Fraser's desire to have a lock-free implementation. As a result of this, their algorithm is somewhat more complicated, and also requires a minimum of $3M + 1$ `CAS` operations, whereas the algorithm of Herlihy et al. [14] requires only $2M + 1$. The authors are unaware of any direct performance comparison of these algorithms, but they believe the obstruction-free one would outperform the lock-free one, particularly in the absence of conflicting `MCAS` operations.

Open Questions

Because transactional memory research has grown out of research into nonblocking data structures, it was long considered mandatory for STM implementations to support the development of nonblocking data structures. Recently, however, a number of researchers have observed that at least the software engineering benefits of transactional memory can be delivered even by a blocking STM. There are ongoing debates whether STM needs to be nonblocking and whether there is a fundamental cost to being nonblocking.

While we agree that blocking STMs are considerably easier to design, and that in many cases a blocking STM is acceptable, this is not always true. Consider, for example, an interrupt handler that shares data with the interrupted thread. The interrupted thread will not run again until the interrupt handler completes, so it is critical that the interrupted thread does not block the interrupt handler. Thus, if using STM is desired to simplify the code for accessing this shared data, the STM *must* be nonblocking. The authors are therefore motivated to continue research aimed at improving nonblocking STMs and to understand what fundamental gap, if any, exists between blocking and nonblocking STMs.

Progress in improving the common-case performance of nonblocking STMs continues [19], and the authors see no reason to believe that nonblocking STMs should not be very competitive with blocking STMs in the common case, i. e., until the system decides that one transaction should not wait for another that is delayed (an option that is not available with blocking STMs).

It is conjectured that indeed a separation between blocking and nonblocking STMs can be proved according to some measure, but that this will not imply significant performance differences in the common case. Indeed results of Attiya et al. [3] show a separation between obstruction-free and blocking algorithms according to a measure that counts the number of distinct base objects accessed by the implementation plus the number of "memory stalls", which measure how often the implementation can encounter contention for a variable from another thread. While this result is interesting, it is not clear that it is useful for deciding whether to implement blocking or obstruction-free objects, because the measure does not account for the time spent waiting by blocking implementations, and thus is biased in their favor. For now, remain optimistic that STMs can be made to be nonblocking without paying a severe performance price in the common case.

Another interesting question, which is open as far as the authors know, is whether there is a fundamental cost to implementing stronger nonblocking progress conditions versus obstruction-freedom. Again, they conjecture that there is. It is known that there is a fundamental difference between obstruction-freedom and lock-freedom in systems that support only reads and writes: It is possible to solve obstruction-free consensus but not lock-free consensus in this model [15]. While this is a fascinating observation, it is mostly irrelevant from a practical standpoint as all modern shared memory multiprocessors support stronger synchronization primitives such as `CAS`, with which it is easy to solve consensus, even wait-free. The interesting question therefore is whether there is a fundamental cost to being lock-free as opposed to obstruction-free in real systems.

To have a real impact on design directions, such results need to address common case performance, or some other measure (perhaps space) that is relevant to everyday use. Many lower bound results establish a separation in worst-case time complexity, which does not necessarily have a direct impact on design decisions, because the worst case may be very rare. So far, efforts to establish a separation according to potentially useful measures have only led to stronger results than we had conjectured were possible. In the authors first attempt [18], they tried to establish a separation in the number of CAS instructions needed in the absence of contention to solve consensus, but found that this was not a very useful measure, as were able to come up with a wait-free implementation that avoids CAS in the absence of contention. The second attempt [6] was to establish a separation according to the *obstruction-free step complexity* measure, which counts the maximum number of steps to complete an operation once the operation encounters no more contention. They knew we could implement obstruction-free DCAS with constant obstruction-free step complexity, and attempt to prove this impossible for lock-free DCAS, but achieved such an algorithm. These experiences suggest that, in addition to their direct advantages, obstruction-free algorithms may provide a useful stepping stone to algorithms with stronger progress properties.

Finally, while a number of contention managers have proved effective for various workloads, it is an open question whether a single contention manager can adapt to be competitive with the best on all workloads, and how close it can come to making optimal contention management decisions. Experience to date suggests that this will be very challenging to achieve. Therefore, as in any system, the first priority should be avoiding contention in the first place. Fortunately, transactional memory has the potential to make this much easier than in lock-based programming models, because it offers the benefits of fine-grained synchronization without the programming complexity that accompanies fine-grained locking schemes.

Cross References

▶ Concurrent Programming, Mutual Exclusion
▶ Linearizability

Recommended Reading

1. Agarwal, A., Cherian, M.: Adaptive backoff synchronization techniques. In: Proceedings of the 16th Annual International Symposium on Computer Architecture, pp. 396–406. ACM Press, New York (1989)
2. Aguilera, M.K., Frolund, S., Hadzilacos, V., Horn, S.L., Toueg, S.: Brief announcement: Abortable and query-abortable objects. In: Proc. 20th Annual International Symposium on Distributed Computing, 2006
3. Attiya, H., Guerraoui, R., Hendler, D., Kouznetsov, P.: Synchronizing without locks is inherently expensive. In: PODC '06: Proceedings of the twenty-fifth Annual ACM Symposium on Prin-

ciples of Distributed Computing, New York, USA, pp. 300–307. ACM Press (2006)
4. Attiya, H., Guerraoui, R., Kouznetsov, P.: Computing with reads and writes in the absence of step contention. In: Proc. 19th Annual International Symposium on Distributed Computing, 2005
5. Damron, P., Fedorova, A., Lev, Y., Luchangco, V., Moir, M., Nussbaum, D.: Hybrid transactional memory. In: Proc. 12th Symposium on Architectural Support for Programming Languages and Operating Systems, 2006
6. Fich, F., Luchangco, V., Moir, M., Shavit, N.: Brief announcement: Obstruction-free step complexity: Lock-free DCAS as an example. In: Proc. 19th Annual International Symposium on Distributed Computing, 2005
7. Fich, F., Luchangco, V., Moir, M., Shavit, N.: Obstruction-free algorithms can be practically wait-free. In: Proc. 19th Annual International Symposium on Distributed Computing, 2005
8. Fraser, K., Harris, T.: Concurrent programming without locks. http://www.cl.cam.ac.uk/netos/papers/2004-cpwl-submission.pdf (2004)
9. Guerraoui, R., Herlihy, M., Pochon, B.: Polymorphic contention management. In: Proc. 19th Annual International Symposium on Distributed Computing, 2005
10. Guerraoui, R., Herlihy, M., Pochon, B.: Toward a theory of transactional contention managers. In: Proc. 24th Annual ACM Symposium on Principles of Distributed Computing, 2005, pp. 258–264
11. Guerraoui, R., Kapalka, M., Kouznetsov, P.: The weakest failure detector to boost obstruction freedom. In: Proc. 20th Annual International Symposium on Distributed Computing, 2006
12. Herlihy, M.: Wait-free synchronization. ACM Trans. Program. Lang. Syst. **13**(1), 124–149 (1991)
13. Herlihy, M.: A methodology for implementing highly concurrent data objects. ACM Trans. Program. Lang. Syst. **15**(5), 745–770 (1993)
14. Herlihy, M., Luchangco, V., Moir, M.: Obstruction-free mechanism for atomic update of multiple non-contiguous locations in shared memory. US Patent Application 20040034673 (2002)
15. Herlihy, M., Luchangco, V., Moir, M.: Obstruction-free synchronization: Double-ended queues as an example. In: Proceedings of the 23rd International Conference on Distributed Computing Systems, 2003
16. Herlihy, M., Luchangco, V., Moir, M., Scherer III., W.: Software transactional memory for supporting dynamic-sized data structures. In: Proc. 22th Annual ACM Symposium on Principles of Distributed Computing, 2003, pp. 92–101
17. Herlihy, M., Moss, J.E.B.: Transactional memory: Architectural support for lock-free data structures. In: Proc. 20th Annual International Symposium on Computer Architecture, 1993, pp. 289–300
18. Luchangco, V., Moir, M., Shavit, N.: On the uncontended complexity of consensus. In: Proc. 17th Annual International Symposium on Distributed Computing, 2005
19. Marathe, V.J., Moir, M.: Toward high performance nonblocking software transactional memory. In: Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming. pp. 227–236, ACM, New York, USA (2008)
20. Marathe, V., Scherer, W., Scott, M.: Adaptive software transactional memory. In: Proc. 19th Annual International Symposium on Distributed Computing, 2005
21. Michael, M., Scott, M.: Nonblocking algorithms and preemption-safe locking on multiprogrammed shared memory multiprocessors. J. Parall. Distrib. Comput. **51**(1), 1–26 (1998)
22. Scherer, W., Scott, M.: Advanced contention management for dynamic software transactional memory. In: Proc. 24th Annual ACM Symposium on Principles of Distributed Computing, 2005
23. Shavit, N., Touitou, D.: Software transactional memory. Distrib. Comput., Special Issue **10**, 99–116 (1997)
24. Treiber, R.: Systems programming: Coping with parallelism. Technical Report RJ5118, IBM Almaden Research Center (1986)

Warehouse Location

▶ Facility Location
▶ Local Search for K-medians and Facility Location

Weighted Bipartite Matching

▶ Assignment Problem

Weighted Caching

▶ Online Paging and Caching

Weighted Connected Dominating Set

2005; Wang, Wang, Li

Yu Wang[1], Weizhao Wang[2], Xiang-Yang Li[3]
[1] Department of Computer Science, University of North Carolina at Charlotte, Charlotte, NC, USA
[2] Google Inc., Irvine, CA, USA
[3] Department of Computer Science, Illinois Institute of Technology, Chicago, IL, USA

Keywords and Synonyms

Minimum weighted connected dominating set

Problem Definition

This problem is concerned with a weighted version of the classical minimum connected dominating set problem. This problem has numerous motivations including wireless networks and distributed systems. Previous work [1,2,4,5,6,14] in wireless networks focuses on designing efficient distributed algorithms to construct the connected dominating set which can be used as the virtual backbone for the network. Most of the proposed methods try to minimize the number of nodes in the backbone (i. e., the number of clusterheads). However, in many applications, minimizing the size of the backbone is not sufficient. For example, in wireless networks different wireless nodes may have different costs for serving as a clusterhead, due to device differences, power capacities, and information loads to be processed. Thus, by assuming each node has a cost to being in the backbone, there is a need to study distributed algorithms for weighted backbone formation. Centralized algorithms to construct a weighted connected dominating set with minimum weight have been studied [3,7,9]. Recently, the work of Wang, Wang, and Li [12,13] proposes an efficient distributed method to construct a weighted backbone with low cost. They proved that the total cost of the constructed backbone is within a small constant factor of the optimum when either the nodes' costs are smooth (i. e. the maximum ratio of costs of adjacent nodes is bounded) or the network maximum node degree is bounded. To the best knowledge of the entry authors, this work is the first to consider this weighted version of minimum connected dominating set problem and provide a distributed approximation algorithm.

Notations

A communication graph $G = (V, E)$ over a set V of wireless nodes has an edge uv between nodes u and v if and only if u and v can communicate directly with each other, i. e., inside the transmission region of each other. Let $d_G(u)$ be the degree of node u in a graph G and Δ be the maximum node degree of all wireless nodes (i. e. $\Delta = \max_{u \in V} d_G(u)$). Each wireless node u has a cost $c(u)$ of being in the backbone. Let $\delta = \max_{ij \in E} c(i)/c(j)$, where ij is the edge between nodes i and j, E is the set of communication links in the wireless network G, and the maximum operation is taken on all pairs of adjacent nodes i and j in G. In other words, δ is the maximum ratio of costs of two adjacent nodes and can be called the *cost smoothness* of the network. When δ is bounded by some small constant, the node costs are *smooth*. When the transmission region of every wireless node is modeled by a unit disk centered at itself, the communication graph is often called a *unit disk graph*, denoted by $UDG(V)$. Such networks are also called *homogeneous networks*.

A subset S of V is a *dominating set* if each node in V is either in S or is adjacent to some node in S. Nodes from S are called dominators, while nodes not in S are called dominatees. A subset B of V is a *connected dominating set* (CDS) if B is a dominating set and B induces a connected subgraph. Consequently, the nodes in B can communicate with each other without using nodes in $V - B$. A dominating set with minimum cardinality is called *minimum dominating set* (MDS). A CDS with minimum cardinality is the *minimum connected dominating set* (MCDS). In the weighted version, assume that each node u has a cost $c(u)$. Then a CDS B is called *weighted connected dominating set* (WCDS). A subset B of V is a *minimum weighted connected dominating set* (MWCDS) if B is a WCDS with minimum total cost. It is well-known that finding either the *minimum connected dominating set* or the *minimum weighted connected dominating set* is a NP-hard problem even when G is a unit disk graph. The work of Wang et al. studies efficient approximation algorithms to construct a low-cost backbone which can approximate the MWCDS problem well. For a given communication graph $G = (V, E, C)$ where V is the set of nodes, E is the edge set, and C is the set of weights for edges, the corresponding minimum weighted connected dominating set problem is as follows.

Problem 1 (Minimum Weighted Connected Dominating Set)
INPUT: *The weighted communication graph* $G = (V, E, C)$.
OUTPUT: *A subset A of V is a minimum weighted connected dominating set, i. e.,* (1) *A is a dominating set;* (2) *A induces a connected subgraph;* (3) *the total cost of A is minimum.*

Another related problem is independent set problem. A subset of nodes in a graph G is an *independent set* if for any pair of nodes, there is no edge between them. It is a *maximal independent set* if no more nodes can be added to it to generate a larger independent set. Clearly, any maximal independent set is a dominating set. It is a *maximum independent set* (MIS) if no other independent set has more nodes. The independence number, denoted as $\alpha(G)$, of a graph G is the size of the MIS of G. The *k-local independence number*, denoted by $\alpha^{[k]}(G)$, is defined as $\alpha^{[k]}(G) = \max_{u \in V} \alpha(G_k(u))$. Here, $G_k(u)$ is the induced graph of G on k-hop neighbors of u (denoted by $N_k(u)$), i. e., $G_k(u)$ is defined on $N_k(u)$, and contains all edges in G with both end-points in $N_k(u)$. It is well-known that for a unit disk graph, $\alpha^{[1]}(UDG) \le 5$ [2] and $\alpha^{[2]}(UDG) \le 18$ [11].

Key Results

Since finding the minimum weighted connected dominating set (MWCDS) is NP-hard, centralized approximation algorithms for MWCDS have been studied [3,7,9]. In [9], Klein and Ravi proposed an approximation algorithm for the node-weighted Steiner tree problem. Their algorithm can be generalized to compute a $O(\log \Delta)$ approximation for MWCDS. Guha and Khuller [7] also studied the approximation algorithms for node-weighted Steiner tree problem and MWCDS. They developed an algorithm for MWCDS with an approximation factor of $(1.35 + \epsilon) \log \Delta$ for any fixed $\epsilon > 0$. Recently, Ambuhl et al. [3] provided a constant approximation algorithm for MWCDS under UDG model. Their approximation ratio is bounded by 89. All these algorithms are centralized algorithms, while the applications in wireless ad hoc networks prefer distributed solutions for MWCDS.

In [12,13], Wang et al. proposed a distributed algorithm that constructs a weighted connected dominating set for a wireless ad hoc network G. Their method has two phases: the first phase (clustering phase, Algorithm 1 in [12,13]) is to find a set of wireless nodes as the dominators (clusterheads) and the second phase (Algorithm 2 in [12,13]) is to find a set of nodes, called *connectors*, to connect these dominators to form the final backbone. Wang et al. proved that the total cost of the constructed backbone is no more than $\min(\alpha^{[2]}(G) \log(\Delta + 1), (\alpha^{[1]}(G) - 1)\delta + 1) + 2\alpha^{[1]}(G)$ times of the optimum solution.

Algorithm 1 first constructs a maximal independent set (MIS) using classical greedy method with the node cost as the selection criterion. For each node v in MIS, it then runs a local greedy set cover method on the *local neighborhood* $N_2(v)$ to find some nodes ($GRDY_v$) to cover all one-hop neighbors of v. If $GRDY_v$ has a total cost smaller than v, then it uses $GRDY_v$ to replace v, which further reduces the cost of MIS. The following theorem of the total cost of this selected set is proved in [12,13].

Theorem 1 *For a network modeled by a graph G, Algorithm 1 (in* [12,13]*) constructs a dominating set whose total cost is no more than* $\min(\alpha^{[2]}(G) \log(\Delta+1), (\alpha^{[1]}(G)-1)\delta+1)$ *times of the optimum.*

Algorithm 2 finds some *connectors* among all the dominatees to connect the dominators into a backbone (CDS). It forms a CDS by finding connectors to connect any pair of dominators u and v if they are connected in the original graph G with at most 3 hops. A distributed algorithm to build a MST then is performed on the CDS. The following theorem of the total cost of these connectors is proved in [12,13].

Theorem 2 *The connectors selected by Algorithm 2 (in* [12,13]*) have a total cost no more than* $2 \cdot \alpha^{[1]}(G)$ *times of the optimum for networks modeled by G.*

Combining Theorem 1 and Theorem 2, the following theorem is the main contributions of the work of Wang et al..

Theorem 3 *For any communication graph G, Algorithm 1 and Algorithm 2 construct a weighted connected dominating set whose total cost is no more than*

$$\min(\alpha^{[2]}(G) \log(\Delta + 1), (\alpha^{[1]}(G) - 1)\delta + 1) + 2\alpha^{[1]}(G)$$

times of the optimum.

Notice that, for homogeneous wireless networks modeled by UDG, it implies that the constructed backbone has a cost no more than $\min(18 \log(\Delta + 1), 4\delta + 1) + 10$ times of the optimum. The advantage of the constructed backbone is that the total cost is small compared with the optimum when either the costs of wireless nodes are smooth, *i. e.*, two neighboring nodes' costs differ by a small constant factor, or the maximum node degree is low.

In term of time complexity, the most time-consuming step in the proposed distributed algorithm is building the MST. In [10], Kuhn et al. gave a lower bound on the distributed time complexity of any distributed algorithm that wants to compute a minimum dominating set in a graph. Essentially, they proved that even for the unconnected and unweighted case, any distributed approximation algorithm with poly-logarithmic approximation guarantee for the problem has to have a time-complexity of at least $\Omega(\log \Delta / \log \log \Delta)$.

Applications

The proposed distributed algorithms for MWCDS can be used in ad hoc networks or distributed system to form a low-cost network backbone for communication application. The cost used as the input of the algorithms could be a *generic* cost, defined by various practical applications. It may represent the *fitness* or *priority* of each node to be a clusterhead. The lower cost means the higher priority. In practice, the cost could represent the power consumption rate of the node if a backbone with small power consumption is needed; the robustness of the node if fault-tolerant backbone is needed; or a function of its security level if a secure backbone is needed; or a combined weight function to integrate various metrics such as traffic load, signal overhead, battery level, and coverage. Therefore, by defining different costs, the proposed low-cost backbone formation algorithms can be used in various practical applications. Beside forming the backbone for routing, the weighted clustering algorithm (Algorithm 1) can also be used in other applications, such as selecting the mobile agents to perform intrusion detection in ad hoc networks [8] (to achieve more robust and power efficient agent selection), or select the rendezvous points to collect and store data in sensor networks [15] (to achieve the energy efficiency and storage balancing).

Open Problems

A number of problems related to the work of Wang, Wang, and Li [12,13] remain open. The proposed method assumes that the nodes are almost-static in a reasonable period of time. However, in some network applications, the network could be highly dynamic (both the topology or the cost could change). Therefore, after the generation of the weighted backbone, the dynamic maintenance of the backbone is also an important issue. It is still unknown how to update the topology efficiently while preserving the approximation quality.

In [12,13], the following assumptions on wireless network model is used: omni-directional antenna, single transmission received by all nodes within the vicinity of the transmitter. The MWCDS problem will become much more complicated if some of these assumptions are relaxed.

Experimental Results

In [12,13], simulations on random networks are conducted to evaluate the performances of the proposed weighted backbone and several backbones built by previous methods. The simulation results confirm the theoretical results.

Cross References

► Connected Dominating Set

Recommended Reading

1. Alzoubi, K., Wan, P.-J., Frieder, O.: New distributed algorithm for connected dominating set in wireless ad hoc networks. In: Proceedings of IEEE 35th Hawaii International Conference on System Sciences (HICSS-35), Hawaii, 7–10 January 2002
2. Alzoubi, K., Li, X.-Y., Wang, Y., Wan, P.-J., Frieder, O.: Geometric spanners for wireless ad hoc networks. IEEE Trans. Parallel Distrib. Process. **14**, 408–421 (2003)
3. Ambuhl, C., Erlebach, T., Mihalak, M., Nunkesser, M.: Constant-factor approximation for minimum-weight (connected) dominating sets in unit disk graphs. In: Proceedings of the 9th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX 2006), Barcelona, 28–30 August 2006, LNCS, vol. 4110, pp. 3–14. Springer, Berlin Heidelberg (2006)
4. Bao, L., Garcia—Aceves, J.J.: Topology management in ad hoc networks. In: Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing, Annapolis, 1–3 June 2003, pp. 129–140. ACM Press, New York (2003)
5. Chatterjee, M., Das, S., Turgut, D.: WCA: A weighted clustering algorithm for mobile ad hoc networks. J. Clust. Comput. **5**, 193–204 (2002)
6. Das, B., Bharghavan, V.: Routing in ad-hoc networks using minimum connected dominating sets. In: Proceedings of IEEE International Conference on on Communications (ICC'97), vol. 1, pp. 376–380. Montreal, 8–12 June 1997
7. Guhaa, S., Khuller, S.: Improved methods for approximating node weighted Steiner trees and connected dominating sets. Inf. Comput. **150**, 57–74 (1999)
8. Kachirski, O., Guha, R.: Intrusion detection using mobile agents in wireless ad hoc networks. In: Proceedings of IEEE Workshop on Knowledge Media Networking, Kyoto, 10–12 July 2002
9. Klein, P., Ravi, R.: A nearly best-possible approximation algorithm for node-weighted Steiner trees. J. Algorithms **19**, 104–115 (1995)
10. Kuhn, F., Moscibroda, T., Wattenhofer, R.: What cannot be computed locally! In: Proceedings of the 23rd ACM Symposium on the Principles of Distributed Computing (PODC), St. John's, July (2004)
11. Li, X.-Y., Wan, P.-J.: Theoretically good distributed CDMA/OVSF code assignment for wireless ad hoc networks. In: Proceedings of 11th Internatioanl Computing and Combinatorics Conference (COCOON), Kunming, 16–19 August 2005
12. Wang, Y., Wang, W., Li, X.-Y.: Efficient distributed low-cost backbone formation for wireless networks. In: Proceedings of 6th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc 2005), Urbana-Champaign, 25–27 May 2005
13. Wang, Y., Wang, W., Li, X.-Y.: Efficient distributed low cost backbone formation for wireless networks. IEEE Trans. Parallel Distrib. Syst. **17**, 681–693 (2006)
14. Wu, J., Li, H.: A dominating-set-based routing scheme in ad hoc wireless networks. The special issue on Wirel. Netw. Telecommun. Systems J. **3**, 63–84 (2001)
15. Zheng, R., He, G., Gupta, I., Sha, L.: Time idexing in sensor networks. In: Proceedings of 1st IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS), Fort Lauderdale, 24–27 October 2004

Weighted Popular Matchings

2006; Mestre

JULIÁN MESTRE
Department of Computer Science, University of Maryland, College Park, MD, USA

Problem Definition

Consider the problem of matching a set of individuals X to a set of items Y where each individual has a weight and a personal preference over the items. The objective is to construct a matching M that is stable in the sense that there is no matching M' such that the weighted majority vote will choose M' over M.

More formally, a bipartite graph (X, Y, E), a weight $w(x) \in R^+$ for each individual $x \in X$, and a rank function $r : E \to \{1, \ldots, |Y|\}$ encoding the individual preferences are given. For every applicant x and items $y_1, y_2 \in Y$ say applicant x prefers y_1 over y_2 if $r(x, y_1) < r(x, y_2)$, and x is indifferent between y_1 and y_2 if $r(x, y_1) = r(x, y_2)$.

The preference lists are said to be strictly ordered if applicants are never indifferent between two items, otherwise the preference lists are said to contain ties.

Let M and M' be two matchings. An applicant x prefers M over M' if x prefers the item he/she gets in M over the item he/she gets in M'. A matching M is *more popular than* M' if the applicants that prefer M over M' outweigh those that prefer M' over M. Finally, a matching M is *weighted popular* if there is no matching M' more popular than M.

In the *weighted popular matching problem* it is necessary to determine if a given instance admits a popular matching, and if so, to produce one. In the *maximum weighted popular matching problem* it is necessary to find a popular matching of maximum cardinality, provided one exists.

Abraham et al. [2] gave the first polynomial time algorithms for the special case of these problems where the weights are uniform. Later, Mestre [8] introduced the weighted variant and developed polynomial time algorithms for it.

Key Results

Theorem 1 *The weighted popular matching and maximum weighted popular matching problems on instances with strictly ordered preferences can be solved in $O(|X| + |E|)$ time.*

Theorem 2 *The weighted popular matching and maximum weighted popular matching problems on instances with arbitrary preferences can be solved in $O(\min\{k\sqrt{|X|}, |X|\}|E|)$ time.*

Both results rely on an alternative easy-to-compute characterization of weighted popular matchings called *well-formed* matchings. It can be shown that every popular matching is well-formed. While in unweighted instances every well-formed matching is popular [2], in weighted instances there may be well-formed matchings that are not popular. These non-popular well-formed matchings can be weeded out by pruning certain bad edges that cannot be part of any popular matching. In other words, the instance can be pruned so that a matching is popular if and only if it is well-formed and is contained in the pruned instance [8].

Applications

Many real-life problems can be modeled using one-sided preferences. For example, the assignment of graduates to training positions [5], families to government-subsidized housing [10], students to projects [9], and Internet rental markets [1] such as Netflix where subscribers are assigned DVDs.

Furthermore, the weighted framework allows one to model the naturally occurring situation in which some subset of users has priority over the rest. For example, an Internet rental site may offer a "premium" subscription plan and promise priority over "regular" subscribers.

Cross References

▶ Ranked Matching
▶ Stable Marriage

Recommended Reading

1. Abraham, D.J., Chen, N., Kumar, V., Mirrokni, V.: Assignment problems in rental markets. In: Proceedings of the 2nd Workshop on Internet and Network Economics, Patras, December 15–17 2006
2. Abraham, D.J., Irving, R.W., Kavitha, T., Mehlhorn, K.: Popular matchings. In: Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 424–432 (2005)
3. Abraham, D.J., Kavitha, T.: Dynamic matching markets and voting paths. In: Proceedings of the 10th Scandinavian Workshop on Algorithm Theory (SWAT), pp. 65–76, Riga, July 6–8 2006
4. Gardenfors, P.: Match making: assignments based on bilateral preferences. Behav. Sci. **20**, 166–173 (1975)
5. Hylland, A., Zeeckhauser, R.: The efficent allocation of individuals to positions. J. Polit. Econ. **87**(2), 293–314 (1979)
6. Mahdian, M.: Random popular matchings. In: Proceedings of the 7th ACM Conference on Electronic Commerce (EC), pp. 238–242 Venice, July 10–14 2006
7. Manlove, D., Sng, C.: Popular matchings in the capacitated house allocation problem. In: Proceedings of the 14th Annual European Symposium on Algorithms (ESA), pp. 492–503 (2006)
8. Mestre, J.: Weighted popular matchings. In: Proceedings of the 16th International Colloquium on Automata, Languages, and Programming (ICALP), pp. 715–726 (2006)
9. Proll, L.G.: A simple method of assigning projects to students. Oper. Res. Q. **23**(23), 195–201 (1972)
10. Yuan, Y.: Residence exchange wanted: a stable residence exchange problem. Eur. J. Oper. Res. **90**, 536–546 (1996)

Weighted Random Sampling

2005; Efraimidis, Spirakis

PAVLOS EFRAIMIDIS[1], PAUL SPIRAKIS[2]
[1] Department of Electrical and Computer Engineering, Democritus University of Thrace, Xanthi, Greece
[2] Department of Computer Engineering and Informatics, Research and Academic Computer Technology Institute, Patras University, Patras, Greece

Keywords and Synonyms

Random number generation; Sampling

Problem Definition

The problem of random sampling without replacement (RS) calls for the selection of m distinct random items out of a population of size n. If all items have the same probability to be selected, the problem is known as uniform RS. Uniform random sampling in one pass is discussed in [1,6,11]. Reservoir-type uniform sampling algorithms over data streams are discussed in [12]. A parallel uniform random sampling algorithm is given in [10]. In weighted random sampling (WRS) the items are weighted and the probability of each item to be selected is determined by its relative weight. WRS can be defined with the following algorithm D:

Algorithm D, a definition of WRS

```
Input: A population V of n weighted items
Output: A set S with a WRS of size $m$
1: For $k = 1$ to $m$ do
2:     Let $p_i(k) = w_i / \sum_{s_j \in V-S} w_j$ be the probability of
       item $v_i$ to be selected in round $k$
3:     Randomly select an item $v_i \in V - S$ and insert it
       into S
4: End-For
```

Problem 1 (WRS)
INPUT: *A population V of n weighted items.*
OUTPUT: *A set S with a weighted random sample.*

The most important algorithms for WRS are the Alias Method, Partial Sum Trees and the Acceptance/Rejection method (see [9] for a summary of WRS algorithms). *None of these algorithms is appropriate for one-pass WRS.* In this work, an algorithm for WRS is presented. The algorithm is simple, very flexible, and solves the WRS problem over data streams. Furthermore, the algorithm admits parallel or distributed implementation. To the best knowledge of the entry authors, this is the first algorithm for WRS over data streams and for WRS in parallel or distributed settings.

Definitions

One-pass WRS is the problem of generating a weighted random sample in one-pass over a population. If additionally the population size is initially unknown (e.g. a data streams), the random sample can be generated with *reservoir sampling* algorithms. These algorithms keep an auxiliary storage, the reservoir, with all items that are candidates for the final sample.

Notation and Assumptions

The item weights are initially unknown, strictly positive reals. The population size is n, the size of the random sample is m and the weight of item v_i is w_i. The function *random*(L, H) generates a uniform random number in (L, H). X denotes a random variable. Infinite precision arithmetic is assumed. Unless otherwise specified, all sampling problems are without replacement. Depending on the context, WRS is used to denote a weighted random sample or the operation of weighted random sampling.

Key Results

All the results with their proofs can be found in [4].

The crux of the WRS approach of this work is given with the following **algorithm A:**

Algorithm A

```
Input: A population V of $n$ weighted items
Output: A WRS of size $m$
1: For each $v_i \in V$, $u_i = random(0,1)$ and $k_i = u_i^{(1/w_i)}$
2: Select the $m$ items with the largest keys $k_i$ as a WRS
```

Theorem 1 *Algorithm A generates a WRS.*

A reservoir-type adaptation of algorithm A is the following **algorithm A-Res**:

Algorithm A with a Reservoir (A-Res)

```
Input: A population V of $n$ weighted items
Output: A reservoir R with a WRS of size $m$
1: The first $m$ items of V are inserted into R
2: For each item $v_i \in R$: Calculate a key $k_i = u_i^{(1/w_i)}$,
   where $u_i = random(0,1)$
3: Repeat Steps 4–7 for $i = m+1, m+2, \ldots, n$
4:     The smallest key in R is the current threshold T
5:     For item $v_i$: Calculate a key $k_i = u_i^{(1/w_i)}$, where
       $u_i = random(0,1)$
6:     If the key $k_i$ is larger than T, then:
7:         The item with the minimum key in R is
           replaced by item $v_i$
```

Algorithm A-Res performs the calculations required by algorithm A and hence by Theorem 1 A-Res generates a WRS. The number of reservoir operations for algorithm A-Res is given by the following Proposition:

Theorem 2 *If A-Res is applied on n weighted items, where the weights $w_i > 0$ are independent random variables with a common continuous distribution, then the expected number of reservoir insertions (without the initial m insertions)* *is:*

$$\sum_{i=m+1}^{n} P\,[\,\textit{item } i \textit{ is inserted into } S\,] = \sum_{i=m+1}^{n} \frac{m}{i} = O\left(m \cdot \log\left(\frac{n}{m}\right)\right).$$

Let S_w be the sum of the weights of the items that will be skipped by A-Res until a new item enters the reservoir. If T_w is the current threshold to enter the reservoir, then S_w is a continuous random variable that follows an exponential distribution. Instead of generating a key for every item, it is possible to generate random jumps that correspond to the sum S_w. Similar techniques have been applied for uniform random sampling (see for example [3]). The following algorithm A-ExpJ is an exponential jumps-type adaptation of algorithm A:

Algorithm A with exponential jumps (A-ExpJ)

```
Input:  A population V of $n$ weighted items
Output:  A reservoir R with a WRS of size $m$
1:  The first $m$ items of V are inserted into R
2:  For each item $v_i \in R$: Calculate a key $k_i = u_i^{(1/w_i)}$,
    where $u_i = random(0, 1)$
3:  The threshold $T_w$ is the minimum key of R
4:  Repeat Steps 5–10 until the population is exhausted
5:      Let $r = random(0, 1)$ and $X_w = \log(r)/\log(T_w)$
6:      From the current item $v_c$ skip items until item $v_i$,
        such that:
7:      $w_c + w_{c+1} + \cdots + w_{i-1} < X_w$
        $\leq w_c + w_{c+1} + \cdots + w_{i-1} + w_i$
8:      The item in R with the minimum key is replaced by
        item $v_i$
9:      Let $t_w = {T_w}^{w_i}$, $r_2 = random(t_w, 1)$ and $v_i$'s key:
        $k_i = {r_2}^{(1/w_i)}$
10:     The new threshold $T_w$ is the new minimum key
        of R
```

Theorem 3 *Algorithm A-ExpJ generates a WRS.*

The number of exponential jumps of A-ExpJ is given by Proposition 2. Hence algorithm A-ExpJ reduces the number of random variates that have to be generated from $O(n)$ (for A-Res) to $O(m \log(n/m))$. Since generating high-quality random variates can be a costly operation this is a significant improvement for the complexity of the sampling algorithm.

Applications

Random sampling is a fundamental problem in computer science with applications in many fields including databases (see [5,9] and the references therein), data mining, and approximation algorithms and randomized algorithms [7]. Consequently, algorithm A for WRS is a general tool that can find applications in the design of randomized algorithms. For example, algorithm A can be used within approximation algorithms for the k-Median [7].

The reservoir based versions of algorithm A, A-Res and A-ExpJ, have very small requirements for auxiliary storage space (m keys organized as a heap) and during the sampling process their reservoir continuously contains a weighted random sample that is valid for the already processed data. This makes the algorithms applicable to the emerging area of algorithms for processing data streams([2,8]).

Algorithms A-Res and A-ExpJ can be used for weighted random sampling with replacement from data streams. In particular, it is possible to generate a weighted random sample with replacement of size k with A-Res or A-ExpJ, by running concurrently, in one pass, k instances of A-Res or A-ExpJ respectively. Each algorithm instance must be executed with a trivial reservoir of size 1. At the end, the union of all reservoirs is a WRS with replacement.

URL to Code

The algorithms presented in this work are easy to implement. An experimental implementation in Java can be found at: http://utopia.duth.gr/~pefraimi/projects/WRS/index.html

Cross References

▶ Online Paging and Caching
▶ Randomization in Distributed Computing

Recommended Reading

1. Ahrens, J.H., Dieter, U.: Sequential random sampling. ACM Trans. Math. Softw. **11**, 157–169 (1985)
2. Babcock, B., Babu, S., Datar, M., Motwani, R., Widom, J.: Models and issues in data stream systems. In: Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, pp. 1–16. ACM Press (2002)
3. Devroye, L.: Non-uniform Random Variate Generation. Springer, New York (1986)
4. Efraimidis, P., Spirakis, P.: Weighted Random Sampling with a reservoir. Inf. Process. Lett. J. **97**(5), 181–185 (2006)
5. Jermaine, C., Pol, A., Arumugam, S.: Online maintenance of very large random samples. In: SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data, New York, pp. 299–310. ACM Press (2004)
6. Knuth, D.: The Art of Computer Programming, vol. 2 : Seminumerical Algorithms, 2nd edn. Addison-Wesley Publishing Company, Reading (1981)

7. Lin, J.-H., Vitter, J.: ϵ-approximations with minimum packing constraint violation. In: 24th ACM STOC, pp. 771–782 (1992)
8. Muthukrishnan, S.: Data streams: Algorithms and applications. Found. Trends Theor. Comput. Sci. **1**, pp.1–126 (2005)
9. Olken, F.: Random Sampling from Databases. Ph. D. thesis, Department of Computer Science, University of California, Berkeley (1993)
10. Rajan, V., Ghosh, R., Gupta, P.: An efficient parallel algorithm for random sampling. Inf. Process. Lett. **30**, 265–268 (1989)
11. Vitter, J.: Faster methods for random sampling. Commun. ACM **27**, 703–718 (1984)
12. Vitter, J.: Random sampling with a reservoir. ACM Trans. Math. Softw. **11**, 37–57 (1985)

Well Separated Pair Decomposition

2003; Gao, Zhang

Jie Gao[1], Li Zhang[2]
[1] Department of Computer Science, Stony Brook University, Stony Brook, NY, USA
[2] HP Labs, Palo Alto, CA, USA

Keywords and Synonyms

Proximity algorithms for growth-restricted metrics

Problem Definition

Well-separated pair decomposition, introduced by Callahan and Kosaraju [3], has found numerous applications in solving proximity problems for points in the Euclidean space. A pair of point sets (A, B) is *c-well-separated* if the distance between A and B is at least c times the diameters of both A and B. A well-separated pair decomposition of a point set consists of a set of well-separated pairs that "cover" all the pairs of distinct points, i. e., any two distinct points belong to the different sets of some pair. Callahan and Kosaraju [3] showed that for any point set in a Euclidean space and for any constant $c \geq 1$, there always exists a c-well-separated pair decomposition (c-WSPD) with linearly many pairs. This fact has been very useful for obtaining nearly linear time algorithms for many problems, such as computing k-nearest neighbors, N-body potential fields, geometric spanners, approximate minimum spanning trees, etc. Well-separated pair decomposition has also been shown to be very useful for obtaining efficient dynamic, parallel, and external memory algorithms.

The definition of well-separated pair decomposition can be naturally extended to any metric space. However, a general metric space may not admit a well-separated pair decomposition with a subquadratic size. Indeed, even for the metric induced by a star tree with unit weight on each edge,[1] any well-separated pair decomposition requires quadratically many pairs. This makes the well-separated pair decomposition useless for such a metric. However, it has been shown that for the unit disk graph metric, there do exist well-separated pair decompositions with almost linear size, and therefore many proximity problems under the unit disk graph metric can be solved efficiently.

Unit-Disk Graphs [4]

Denote by $\mathrm{d}(\cdot, \cdot)$ the Euclidean metric. For a set of points S in the plane, the unit-disk graph $I(S) = (S, E)$ is defined to be the weighted graph where an edge $e = (p, q)$ is in the graph if $\mathrm{d}(p, q) \leq 1$, and the weight of e is $\mathrm{d}(p, q)$. Likewise, one can define the unit-ball graph for points in higher dimensions.

Unit-disk graphs have been used extensively to model the communication or influence between objects [9,12] and have been studied in many different contexts [4,10]. For an example, wireless ad hoc networks can be modeled by unit-disk graphs [6], as two wireless nodes can directly communicate with each other only if they are within a certain distance. In unsupervised learning, for a dense sampling of points from some unknown manifold, the length of the shortest path on the unit-ball graph is a good approximation of the geodesic distance on the underlying (unknown) manifold if the radius is chosen appropriately [14,5]. By using well-separated pair decomposition, one can encode the all-pairs distances approximately by a compact data structure that supports approximate distance queries in $O(1)$ time.

Metric Space

Suppose that (S, π) is a metric space where S is a set of elements and π the distance function defined on $S \times S$. For any subset $S_1 \subseteq S$, the *diameter* $D_\pi(S_1)$ (or $D(S_1)$ when π is clear from the context) of S is defined to be $\max_{s_1, s_2 \in S_1} \pi(s_1, s_2)$. The *distance* $\pi(S_1, S_2)$ between two sets $S_1, S_2 \subseteq S$ is defined to be $\min_{s_1 \in S_1, s_2 \in S_2} \pi(s_1, s_2)$.

Well-Separated Pair Decomposition

For a metric space (S, π), two nonempty subsets $S_1, S_2 \subseteq S$ are called *c-well-separated* if $\pi(S_1, S_2) \geq c \cdot \max(D_\pi(S_1), D_\pi(S_2))$.

Following the definition in [3], for any two sets A and B, a set of pairs $\mathcal{P} = \{P_1, P_2, \dots, P_m\}$, where $P_i = (A_i, B_i)$, is called a *pair decomposition* of (A, B) (or of A if $A = B$) if

[1] A metric induced by a graph (with positive edge weights) is the shortest-path distance metric of the graph.

- For all the i's, $A_i \subseteq A$, and $B_i \subseteq B$.
- $A_i \cap B_i = \emptyset$.
- For any two elements $a \in A$ and $b \in B$, there exists a unique i such that $a \in A_i$, and $b \in B_i$. Call (a, b) is *covered* by the pair (A_i, B_i).

If in addition, every pair in $\mathcal{P}$ is c-well-separated, $\mathcal{P}$ is called a *c-well-separated pair decomposition* (or c-WSPD for short). Clearly, any metric space admits a c-WSPD with quadratic size by using the trivial family that contains all the pairwise elements.

Key Results

In [7], it was shown that for the metric induced by the unit-disk graph on n points and for any constant $c \geq 1$, there does exist a c-WSPD with $O(n \log n)$ pairs, and such a decomposition can be computed in $O(n \log n)$ time. It was also shown that the bounds can be extended to higher dimensions. The following theorems state the key results for two and higher dimensions.

Theorem 1 *For any set S of n points in the plane and any $c \geq 1$, there exists a c-WSPD $\mathcal{P}$ of S under the unit disk graph metric where $\mathcal{P}$ contains $O(c^4 n \log n)$ pairs and can be computed in $O(c^4 n \log n)$ time.*

Theorem 2 *For any set S of n points in $\mathbb{R}^k$, for $k \geq 3$, and for any constant $c \geq 1$, there exists a c-WSPD $\mathcal{P}$ of S under the unit ball graph metric where $\mathcal{P}$ contains $O(n^{2-2/k})$ pairs and can be constructed in $O(n^{4/3}\ \mathrm{polylog}\ n)$ time for $k = 3$ and in $O(n^{2-2/k})$ time for $k \geq 4$.*

The difficulty in obtaining a well-separated pair decomposition for the unit disk graph metric is that two points that are close in space are not necessarily close under the graph metric. The above bounds are first shown for the point set with constant-bounded density, i. e., a point set where any unit disk covers only a constant number of points in the set. The upper bound on the number of pairs is obtained by using a packing argument similar to the one used in [8].

For a point set with unbounded density, one applies a clustering technique similar to the one used in [6] to the point set and obtains a set of "clusterheads" with a bounded density. Then the result for bounded density is applied to those clusterheads. Finally, the well-separated pair decomposition is obtained by combining the well-separated pair decomposition for the bounded density point sets and for the Euclidean metric. The number of pairs is dominated by the number of pairs constructed for a constant density set, which is in turn dominated by the bound given by the packing argument. It has been shown that the bounds on the number of pairs is tight for $k \geq 3$.

Applications

For a pair of well-separated sets, the distance between two points from different sets can be approximated by the "distance" between the two sets or the distance between any pair of points in different sets. In other words, a well-separated pair decomposition can be thought of as a compressed representation to approximate the $\Theta(n^2)$ pairwise distances. Many problems that require the pairwise distances to be checked can therefore be approximately solved by examining those distances between the well-separated pairs of sets. When the size of the well-separated pair decomposition is subquadratic, it often results in more efficient algorithms than examining all the pairwise distances. Indeed, this is the intuition behind many applications of the geometric well-separated pair decomposition. By using the same intuition, one can apply the well-separated pair decomposition in several proximity problems under the unit disk graph metric.

Suppose that (S, d) is a metric space. Let $S_1 \subseteq S$. Consider the following natural proximity problems.

- **Furthest neighbor, diameter, center.** The furthest neighbor of $p \in S_1$ is the point in S_1 that maximizes the distance to p. Related problems include computing the *diameter*, the maximum pairwise shortest distance for points in S_1, and the *center*, the point that minimizes the maximum distance to all the other points.
- **Nearest neighbor, closest pair.** The nearest neighbor of $p \in S_1$ is the point in S_1 with the minimum distance to p. Related problems include computing the *closest pair*, the pair with the minimum shortest distance, and the *bichromatic closest pair*, the pair that minimizes the distance between points from two different sets.
- **Median.** The median of S is the point in S that minimizes the average (or total) distance to all the other points.
- **Stretch factor.** For a graph G defined on S, its stretch factor with respect to the unit disk graph metric is defined to be the maximum ratio $\pi_G(p, q)/\pi(p, q)$, where π_G, π are the distances induced by G and by the unit-disk graph, respectively.

All the above problems can be solved or approximated efficiently for points in the Euclidean space. However, for the metric induced by a graph, even for planar graphs, very little is known besides solving the expensive all-pairs shortest-path problem. For computing the diameter, there is a simple linear-time method that achieves a 2-approximation[2] and a 4/3-approximate algorithm with running time $O(m\sqrt{n \log n} + n^2 \log n)$, for a graph with n vertices and m edges, by Aingworth et al. [1].

By using the well-separated pair decomposition, Gao and Zhang [7] showed that one can obtain better approximation algorithms for the above proximity problems for the unit disk graph metric. Specifically, one can obtain almost linear-time algorithms for computing the 2.42-approximation and $O(n\sqrt{n \log n}/\varepsilon^3)$ time algorithms for computing the $(1+\varepsilon)$-approximation for any $\varepsilon > 0$. In addition, the well-separated pair decomposition can be used to obtain an $O(n \log n/\varepsilon^4)$ space distance oracle so that any $(1+\varepsilon)$ distance query in the unit-disk graph can be answered in $O(1)$ time.

The bottleneck of the above algorithms turns out to be computing the approximation of the shortest path distances between $O(n \log n)$ pairs. The algorithm in [7] only constructs well-separated pair decompositions without computing a good approximation of the distances. The approximation ratio and the running time are dominated by that of the approximation algorithms used to estimate the distance between each pair in the well-separated pair decomposition. Once the distance estimation has been made, the rest of the computation only takes almost linear time.

For a general graph, it is unknown whether $O(n \log n)$ pairs shortest-path distances can be computed significantly faster than all-pairs shortest-path distances. For a planar graph, one can compute the $O(n \log n)$ pairs shortest-path distances in $O(n\sqrt{n \log n})$ time by using separators with $O(\sqrt{n})$ size [2]. This method extends to the unit-disk graph with constant bounded density since such graphs enjoy a separator property similar to that of planar graphs [13]. As for approximation, Thorup [15] recently discovered an algorithm for planar graphs that can answer any $(1+\varepsilon)$-shortest-distance query in $O(1/\varepsilon)$ time after almost linear time preprocessing. Unfortunately, Thorup's algorithm uses balanced shortest-path separators in planar graphs which do not obviously extend to the unit-disk graphs. On the other hand, it is known that there does exist a planar 2.42-spanner for a unit-disk graph [11]. By applying Thorup's algorithm to that planar spanner, one can compute the 2.42-approximate shortest-path distance for $O(n \log n)$ pairs in almost linear time.

[2]Select an arbitrary node v and compute the shortest-path tree rooted at v. Suppose that the furthest node from v is distance D away. Then the diameter of the graph is no longer than $2D$, by triangle inequality.

Open Problems

The most notable open problem is the gap between $\Omega(n)$ and $O(n \log n)$ on the number of pairs needed in the plane. Also, the time bound for $(1+\varepsilon)$-approximation is still about $\widetilde{O}(n\sqrt{n})$ due to the lack of efficient methods for computing the $(1+\varepsilon)$-approximate shortest path distances between $O(n)$ pairs of points. Any improvement to the algorithm for that problem will immediately lead to improvement to all the $(1+\varepsilon)$-approximate algorithms presented in this chapter.

Cross References

► Applications of Geometric Spanner Networks
► Separators in Graphs
► Sparse Graph Spanners
► Well Separated Pair Decomposition for Unit–Disk Graph

Recommended Reading

1. Aingworth, D., Chekuri, C., Motwani, R.: Fast estimation of diameter and shortest paths (without matrix multiplication). In: Proc. 7th ACM-SIAM Symposium on Discrete Algorithms, 1996, pp. 547–553
2. Arikati, S.R., Chen, D.Z., Chew, L.P., Das, G., Smid, M.H.M., Zaroliagis, C.D: Planar spanners and approximate shortest path queries among obstacles in the plane. In: Díaz, J., Serna, M. (eds.) Proc. of 4th Annual European Symposium on Algorithms, 1996, pp. 514–528
3. Callahan, P.B., Kosaraju, S. R.: A decomposition of multidimensional point sets with applications to *k*-nearest-neighbors and *n*-body potential fields. J. ACM **42**, 67–90 (1995)
4. Clark, B.N., Colbourn, C.J., Johnson, D.S.: Unit disk graphs. Discret. Math. **86**, 165–177 (1990)
5. Fischl, B., Sereno, M., Dale, A.: Cortical surface-based analysis II: Inflation, flattening, and a surface-based coordinate system. NeuroImage **9**, 195–207 (1999)
6. Gao, J., Guibas, L.J., Hershberger, J., Zhang, L., Zhu, A.: Geometric spanners for routing in mobile networks. IEEE J. Sel. Areas Commun. Wirel. Ad Hoc Netw. (J-SAC), **23**(1), 174–185 (2005)
7. Gao, J., Zhang, L.: Well-separated pair decomposition for the unit-disk graph metric and its applications. In: Proc. of 35th ACM Symposium on Theory of Computing (STOC'03), 2003, pp. 483–492
8. Guibas, L., Ngyuen, A., Russel, D., Zhang, L.: Collision detection for deforming necklaces. In: Proc. 18th ACM Symposium on Computational Geometry, 2002, pp. 33–42
9. Hale, W. K.: Frequency assignment: Theory and applications. Proc. IEEE. **68**(12), 1497–1513 (1980)
10. H.B.H. III, Marathe, M.V., Radhakrishnan, V., Ravi, S.S., Rosenkrantz, D.J., Stearns, R.E.: NC-approximation schemes for NP- and PSPACE-hard problems for geometric graphs. J. Algorithms **26**(2), 238–274 (1998)
11. Li, X.Y., Calinescu, G., Wan, P.J.: Distributed Construction of a Planar Spanner and Routing for Ad Hoc Wireless Networks. In: IEEE INFOCOM 2002, New York, NY, 23–27 June 2002

12. Mead, C.A., Conway, L.: Introduction to VLSI Systems. Addison-Wesley, (1980)
13. Miller, G.L., Teng, S.H., Vavasis, S.A.: An unified geometric approach to graph separators. In: Proc. 32nd Annu. IEEE Sympos. Found. Comput. Sci. 1991, pp. 538–547
14. Tenenbaum, J., de Silva, V., Langford, J.: A global geometric framework for nonlinear dimensionality reduction. Science **290**, 22 (2000)
15. Thorup, M.: Compact oracles for reachability and approximate distances in planar digraphs. In: Proc. 42nd IEEE Symposium on Foundations of Computer Science, 2001, pp. 242–251

Well Separated Pair Decomposition for Unit–Disk Graph

1995; Callahan, Kosaraju

ROLF KLEIN
Institute of Computer Science, University of Bonn, Bonn, Germany

Keywords and Synonyms

Clustering

Problem Definition

Notations

Given a finite point set A in $\mathbb{R}^d$, its *bounding box* $R(A)$ is the d-dimensional hyper-rectangle $[a_1, b_1] \times [a_2, b_2] \times \ldots \times [a_d, b_d]$ that contains A and has minimum extension in each dimension.

Two point sets A, B are said to be *well-separated* with respect to a separation parameter $s > 0$ if there exist a real number $r > 0$ and two d-dimensional spheres C_A and C_B of radius r each, such that the following properties are fulfilled.

1. $C_A \cap C_B = \emptyset$
2. C_A contains the bounding box $R(A)$ of A
3. C_B contains the bounding box $R(B)$ of B
4. $|C_A C_B| \geq s \cdot r$.

Here $|C_A C_B|$ denotes the smallest Euclidean distance between two points of C_A and C_B, respectively. An example is depicted in Fig. 1. Given the bounding boxes $R(A)$, $R(B)$, it takes time only $O(d)$ to test if A and B are well-separated with respect to s.

Two points of the same set, A or B, have a Euclidean distance at most $2/s$ times the distance any pair $(a, b) \in A \times B$ can have. Also, any two such pairs $(a, b), (a', b')$ differ in their distances $|a - b|, |a' - b'|$ by a factor of at most $1 + 4/s$.

Given a set S of n points in $\mathbb{R}^d$, a *well-separated pair decomposition* of S with respect to separation parameter s is a sequence $(A_1, B_1), (A_2, B_2), \ldots, (A_m, B_m)$ where

1. $A_i, B_i \subset S$, for $i = 1 \ldots m$
2. A_i and B_i are well-separated with respect to s, for $i = 1 \ldots m$
3. for all points $a, b \in S$, $a \neq b$, there exists a unique index i in $1 \ldots m$ such that $a \in A_i$ and $b \in B_i$, or $b \in A_i$ and $a \in B_i$ hold

Obviously, each set $S = \{s_1, \ldots, s_n\}$ possesses a well-separated pair decomposition. One can simply use all singleton pairs $(\{s_i\}, \{s_j\})$ where $i < j$. The question is if decompositions consisting of fewer than $O(n^2)$ many pairs exist, and how to construct them efficiently.

Key Results

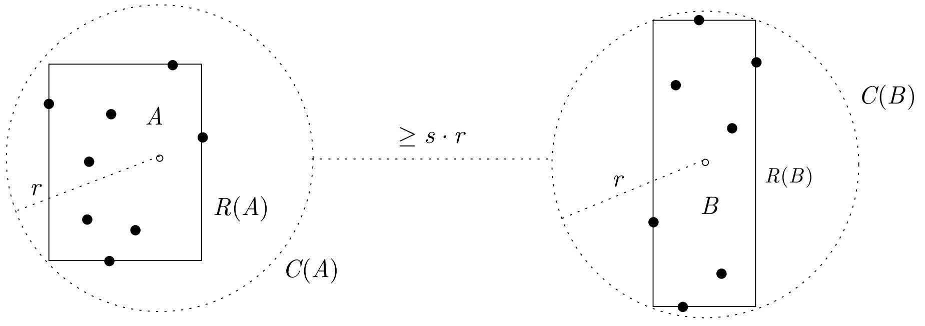
In fact, the following result has been shown by Callahan and Kosaraju [1,2].

Theorem 1 *Given a set S of n points in $\mathbb{R}^d$ and a separation parameter s, there exists a well-separated pair decomposition of S with respect to s, that consists of $O(s^d d^{d/2} n)$ many pairs (A_i, B_i). It can be constructed in time $O(dn \log n + s^d d^{d/2+1} n)$.*

Thus, if dimension d and separation parameter s are fixed – which is the case in many applications – then the number of pairs is in $O(n)$, and the decomposition can be computed in time $O(n \log n)$.

The main tool in constructing the well-separated pair decomposition is the *split tree* $T(S)$ of S. The root, r, of $T(S)$ contains the bounding box $R(S)$ of S. Its two child nodes are obtained by cutting through the middle of the longest dimension of $R(S)$, using an orthogonal hyperplane. It splits S into two subsets S_a, S_b, whose bounding boxes $R(S_a)$ and $R(S_b)$ are stored at the two children a and b of root r. This process continues until only one point of S remains in each subset. These singleton sets form the leaves of $T(S)$. Clearly, the split tree $T(S)$ contains $O(n)$ many nodes. It need not be balanced, but it can be constructed in time $O(dn \log n)$.

A well-separated pair decomposition of S, with respect to a given separation parameter s, can now be obtained from $T(S)$ in the following way. For each internal node of $T(S)$ with children v and w the following recursive procedure FindPairs(v,w) is called. If S_v and S_w are well-separated then the pair (S_v, S_w) is reported. Otherwise, one may assume that the longest dimension of $R(S_v)$ exceeds in length the longest dimension of $R(S_w)$, and that v_l, v_r are the child nodes of v in $T(S)$. Then, FindPairs(v_l, w) and FindPairs(v_r, w) are invoked.



Well Separated Pair Decomposition for Unit–Disk Graph, Figure 1
The sets A, B are well-separated with respect to s

The total number of procedure calls is bounded by the number of well-separated pairs reported, which can be shown to be in $O(s^d d^{d/2} n)$ by a packing argument. However, the total size of all sets A_i, B_i in the decomposition is in general quadratic in n.

Applications

From now on the dimension d is assumed to be a constant. The well-separated pair decomposition can be used in efficiently solving proximity problems for points in $\mathbb{R}^d$.

Theorem 2 *Let S be a set of n points in $\mathbb{R}^d$. Then a closest pair in S can be found in optimal time $O(n \log n)$.*

Indeed, let $q \in S$ be a nearest neighbor of $p \in S$. One can construct a well-separated pair decomposition with separation parameter $s > 2$ in time $O(n \log n)$, and let (A_i, B_i) be the pair where $p \in A_i$ and $q \in B_i$. If there were another point p' of S in A_i, one would obtain $|pp'| \leq 2/s \cdot |pq| < |pq|$, which is impossible. Hence, A_i is a singleton set. If (p, q) is a closest pair in S then B_i must be singleton, too. Therefore, a closest pair can be found by inspecting all singleton pairs among the $O(n)$ many pairs of the well-separated pair decomposition.

With more effort, the following generalization can been shown.

Theorem 3 *Let S be a set of n points in $\mathbb{R}^d$, and let $k \leq n$. Then for each $p \in S$ its k nearest neighbors in S can be computed in total time $O(n \log n + nk)$. In particular, for each point in S can a nearest neighbor in S be computed in optimal time $O(n \log n)$.*

In dimension $d = 2$ one would typically use the *Voronoi diagram* for solving these problems. But as the complexity of the Voronoi diagram of n points can be as large as $n^{\lfloor d/2 \rfloor}$, the well-separated pair decomposition is much more convenient to use in higher dimensions.

A major application of the well-separated pair decomposition is the construction of good *spanners* for a given point set S. A spanner of S of dilation t is a geometric network N with vertex set S such that, for any two vertices $p, q \in S$, the Euclidean length of a shortest path connecting p and q in N is at most t times the Euclidean distance $|pq|$.

Theorem 4 *Let S be a set of n points in $\mathbb{R}^d$, and let $t>1$. Then a spanner of S of dilation t containing $O(s^d\ n)$ edges can be constructed in time $O\ (s^d n + n \log n)$, where $s = 4(t+1)(t-1)$.*

Indeed, if one edge (a_i, b_i) is chosen from each pair (A_i, B_i) of a well-separated pair decomposition of S with respect to s, these edges form a t-spanner of S, as can be shown by induction on the rank of each pair $(p, q) \in S^2$ in the list of all such pairs, sorted by distance.

Since spanners have many interesting applications of their own, several articles of this encyclopedia are devoted to this topic.

Open Problems

An important open question is which metric spaces admit well-separated pair decompositions. It is easy to see that the packing arguments used in the Euclidean case carry over to the case of convex distance functions in $\mathbb{R}^d$. More generally, Talwar [6] has shown how to compute well-separated pair decompositions for point sets of bounded aspect ratio in metric spaces of bounded doubling dimension.

On the other hand, for the metric induced by a disk graph in $\mathbb{R}^2$, a quadratic number of pairs may be necessary in the well-separated pair decomposition. (In a disk graph, each point $p \in S$ is center of a disk D_p of radius r_p. Two points p, q are connected by an edge if and only if $D_p \cap D_q \neq \emptyset$. The metric is defined by Euclidean shortest path length in the resulting graph. If this graph is a star with rays of identical length, a well-separated pair decomposition with respect to $s > 4$ must consist of singleton pairs.) Even for a unit disk graph, $\Omega(n^{2-2/d})$ many pairs may be necessary for points in $\mathbb{R}^d$, as Gao and Zhang [4] have shown.

Cross References

▶ Applications of Geometric Spanner Networks
▶ Geometric Spanners
▶ Planar Geometric Spanners

Recommended Reading

1. Callahan, P.: Dealing with Higher Dimensions: The Well-Separated Pair Decomposition and Its Applications. Ph. D. Thesis, The Johns Hopkins University, USA (1995)
2. Callahan, P.B., Kosaraju, S.R.: A Decomposition of Multidimensional Point Sets with Applications to k-Nearest Neighbors and n-Body Potential Fields. J. ACM **42**(1), 67–90 (1995)
3. Eppstein, D.: Spanning Trees and Spanners. In: Sack, J.R., Urrutia, J. (eds.) Handbook of Computational Geometry, pp. 425–461. Elsevier, Amsterdam (1999)
4. Ghao, J., Zhang, L.: Well-Separated Pair Decomposition for the Unit Disk Graph Metric and its Applications. SIAM J. Comput. **35**(1), 151–169 (2005)
5. Narasimhan, G., Smid, M.: Geometric Spanner Networks. Cambridge University Press, New York (2007)
6. Talwar, K.: Bypassing the Embedding: Approximation Schemes and Compact Representations for Low Dimensional Metrics. In: Proceedings of the thirty-sixth Annual ACM Symposium on Theory of Computing (STOC'04), pp. 281–290 (2004)

Whole Genome Assemble

▶ Multiplex PCR for Gap Closing (Whole-genome Assembly)

Wireless Networks

▶ Broadcasting in Geometric Radio Networks
▶ Deterministic Broadcasting in Radio Networks
▶ Randomized Gossiping in Radio Networks

Wire Sizing

1999; Chu, Wong

Chris Chu
Department of Electrical and Computer Engineering, Iowa State University, Ames, IA, USA

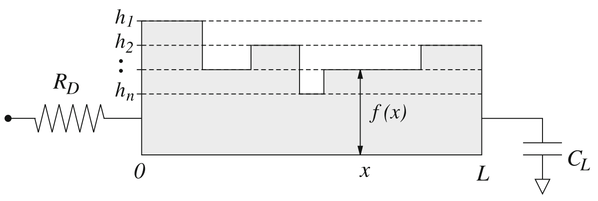
Keywords and Synonyms

Wire tapering

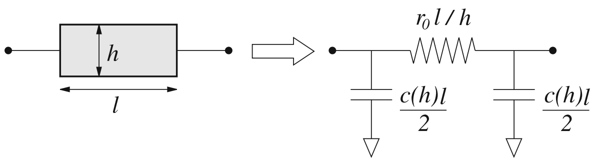
Problem Definition

The problem is about minimizing the delay of an interconnect wire in a Very Large Scale Integration (VLSI) circuit by changing (i. e., sizing) the width of the wire. The delay of interconnect wire has become a dominant factor in determining VLSI circuit performance for advanced VLSI technology. Wire sizing has been shown to be an effective technique to minimize the interconnect delay. The work of Chu and Wong [5] shows that the wire sizing problem can be transformed into a convex quadratic program. This quadratic programming approach is very efficient and can be naturally extended to simultaneously consider buffer insertion, which is another popular interconnect delay minimization technique. Previous approaches apply either a dynamic programming approach [13], which is computationally more expensive, or an iterative greedy approach [2,7], which is hard to combine with buffer insertion.

The wire sizing problem is formulated as follows and is illustrated in Fig. 1. Consider a wire of length L. The wire is connecting a driver with driver resistance R_D to a load with load capacitance C_L. In addition, there is a set $H = \{h_1, \ldots, h_n\}$ of n wire widths allowed by the fabrication technology. Assume $h_1 > \cdots > h_n$. The wire sizing problem is to determine the wire width function $f(x) : [0, L] \to H$ so that the delay for a signal to travel from the driver through the wire to the load is minimized.



Wire Sizing, Figure 1
Illustration of the wire sizing problem



Wire Sizing, Figure 2
The model of a wire segment of length l and width h by a π-type RC circuit

As in most previous works on wire sizing, the work of Chu and Wong uses the Elmore delay model to compute the delay. The Elmore delay model is a delay model for RC circuits (i. e., circuits consisting of resistors and capacitors). The Elmore delay for a signal path is equal to the sum of the delays associated with all resistors along the path, where the delay associated with each resistor is equal to its resistance times its total downstream capacitance. For a wire segment of length l and width h, its resistance is $r_0 l/h$ and its capacitance is $c(h)l$, where r_0 is the wire sheet resistance and $c(h)$ is the unit length wire capacitance. $c(h)$ is an increasing function in practice. The wire segment can be modeled as a π-type RC circuit as shown in Fig. 2.

Key Results

Lemma 1 *The optimal wire width function $f(x)$ is a monotonically decreasing function.*

Lemma 1 above can be used to greatly simplify the wire sizing problem. It implies that an optimally-sized wire can be divided into n segments such that the width of ith segment is h_i. The length of each segment is to be determined. The simplified problem is illustrated in Fig. 3.

Lemma 2 *For the wire in Fig. 3, the Elmore delay is*

$$D = \frac{1}{2}\boldsymbol{l}^{\mathrm{T}}\boldsymbol{\Phi}\,\boldsymbol{l} + \boldsymbol{\rho}^{\mathrm{T}}\boldsymbol{l} + R_D C_L$$

where

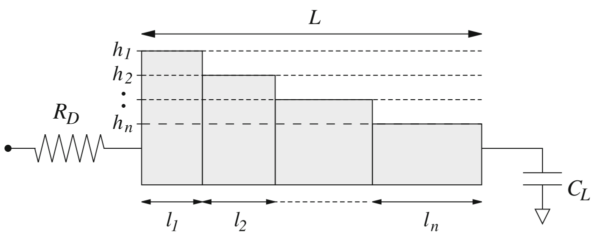
$$\boldsymbol{\Phi} = \begin{pmatrix} c(h_1)r_0/h_1 & c(h_2)r_0/h_1 & c(h_3)r_0/h_1 & \cdots & c(h_n)r_0/h_1 \\ c(h_2)r_0/h_1 & c(h_2)r_0/h_2 & c(h_3)r_0/h_2 & \cdots & c(h_n)r_0/h_2 \\ c(h_3)r_0/h_1 & c(h_3)r_0/h_2 & c(h_3)r_0/h_3 & \cdots & c(h_n)r_0/h_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c(h_n)r_0/h_1 & c(h_n)r_0/h_2 & c(h_n)r_0/h_3 & \cdots & c(h_n)r_0/h_n \end{pmatrix}$$

$$\boldsymbol{\rho} = \begin{pmatrix} R_D c(h_1) + C_L r_0/h_1 \\ R_D c(h_2) + C_L r_0/h_2 \\ R_D c(h_3) + C_L r_0/h_3 \\ \vdots \\ R_D c(h_n) + C_L r_0/h_n \end{pmatrix} \textit{ and } \boldsymbol{l} = \begin{pmatrix} l_1 \\ l_2 \\ l_3 \\ \vdots \\ l_n \end{pmatrix}.$$

So the wire sizing problem can be written as the following quadratic program:

$$\mathcal{WS}: \quad \begin{array}{ll} \text{minimize} & \frac{1}{2}\boldsymbol{l}^{\mathrm{T}}\boldsymbol{\Phi}\,\boldsymbol{l} + \boldsymbol{\rho}^{\mathrm{T}}\boldsymbol{l} \\ \text{subject to} & l_1 + \cdots + l_n = L \\ & l_i \geq 0 \text{ for } 1 \leq i \leq n\,. \end{array}$$

Quadratic programming is NP-hard in general. In order to solve $\mathcal{WS}$ efficiently, some properties of the Hessian matrix $\boldsymbol{\Phi}$ are explored.



Wire Sizing, Figure 3
Illustration of the simplified wire sizing problem

Definition 1 (Symmetric Decomposable Matrix) Let $\boldsymbol{Q} = (q_{ij})$ be an $n \times n$ symmetric matrix. If for some $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)^{\mathrm{T}}$ and $\boldsymbol{v} = (v_1, \ldots, v_n)^{\mathrm{T}}$ such that $0 < \alpha_1 < \cdots < \alpha_n$, $q_{ij} = q_{ji} = \alpha_i v_i v_j$ for $i \leq j$, then $\boldsymbol{Q}$ is called a symmetric decomposable matrix. Let $\boldsymbol{Q}$ be denoted as $SDM(\boldsymbol{\alpha}, \boldsymbol{v})$.

Lemma 3 *If $\boldsymbol{Q}$ is symmetric decomposable, then $\boldsymbol{Q}$ is positive definite.*

Lemma 4 *$\boldsymbol{\Phi}$ in $\mathcal{WS}$ is symmetric decomposable.*

Lemma 3 and Lemma 4 imply that the Hessian matrix $\boldsymbol{\Phi}$ of $\mathcal{WS}$ is positive definite. Hence, the problem $\mathcal{WS}$ is a *convex* quadratic program and is solvable in polynomial time [12].

The work of Chu and Wong proposes to solve $\mathcal{WS}$ by active set method. The active set method transforms a problem with some inequality constraints into a sequence of problems with only equality constraints. The method stops when the solution of the transformed problem satisfies both the feasibility and optimality conditions of the original problem. For the problem $\mathcal{WS}$, the active set method keeps track of an active set $\mathcal{A}$ in each iteration. The method sets $l_j = 0$ for all $j \in \mathcal{A}$ and ignores the constraints $l_j \geq 0$ for all $j \notin \mathcal{A}$. If $\{j_1, \ldots, j_r\} = \{1, \ldots, n\} - \mathcal{A}$, then $\mathcal{WS}$ is transformed into the following equality-constrained wire sizing problem:

$$\mathcal{ECWS}: \quad \begin{array}{ll} \text{minimize} & \frac{1}{2}\boldsymbol{l}_{\mathcal{A}}^{\mathrm{T}}\boldsymbol{\Phi}_{\mathcal{A}}\boldsymbol{l}_{\mathcal{A}} + \boldsymbol{\rho}_{\mathcal{A}}^{\mathrm{T}}\boldsymbol{l}_{\mathcal{A}} \\ \text{subject to} & \boldsymbol{\Gamma}_{\mathcal{A}}\boldsymbol{l}_{\mathcal{A}} = L \end{array}$$

where $\boldsymbol{l}_{\mathcal{A}} = (l_{j_1}, \ldots, l_{j_r})^{\mathrm{T}}$, $\boldsymbol{\Gamma}_{\mathcal{A}} = (1\ 1\ \cdots\ 1)$, $\boldsymbol{\rho}_{\mathcal{A}} = (R_D c(h_{j_1}) + C_L r_0/h_{j_1}, \ldots, R_D c(h_{j_r}) + C_L r_0/h_{j_r})^{\mathrm{T}}$, and $\boldsymbol{\Phi}_{\mathcal{A}}$ is the symmetric decomposable matrix corresponding to $\mathcal{A}$ (i. e., $\boldsymbol{\Phi}_{\mathcal{A}} = SDM(\boldsymbol{\alpha}_{\mathcal{A}}, \boldsymbol{v}_{\mathcal{A}})$ with $\boldsymbol{\alpha}_{\mathcal{A}} = (r_0/c(h_{j_1})h_{j_1}, \ldots, r_0/c(h_{j_r})h_{j_r})^{\mathrm{T}}$ and $\boldsymbol{v}_{\mathcal{A}} = (c(h_{j_1}), \ldots, c(h_{j_r}))^{\mathrm{T}}$).

Lemma 5 *The solution of $\mathcal{ECWS}$ is:*

$$\begin{array}{lll} \lambda_{\mathcal{A}} & = & -(\boldsymbol{\Gamma}_{\mathcal{A}}\boldsymbol{\Phi}_{\mathcal{A}}^{-1}\boldsymbol{\Gamma}_{\mathcal{A}}^{\mathrm{T}})^{-1}(\boldsymbol{\Gamma}_{\mathcal{A}}\boldsymbol{\Phi}_{\mathcal{A}}^{-1}\boldsymbol{\rho}_{\mathcal{A}} + L) \\ \boldsymbol{l}_{\mathcal{A}} & = & -\boldsymbol{\Phi}_{\mathcal{A}}^{-1}\boldsymbol{\Gamma}_{\mathcal{A}}^{\mathrm{T}}\lambda_{\mathcal{A}} - \boldsymbol{\Phi}_{\mathcal{A}}^{-1}\boldsymbol{\rho}_{\mathcal{A}}\,. \end{array}$$

Lemma 6 *If $\mathbf{Q}$ is symmetric decomposable, then $\mathbf{Q}^{-1}$ is tridiagonal. In particular, if $\mathbf{Q} = SDM(\boldsymbol{\alpha}, \mathbf{v})$, then $\mathbf{Q}^{-1} = (\theta_{ij})$ where $\theta_{ii} = 1/(\alpha_i - \alpha_{i-1})v_i^2 + 1/(\alpha_{i+1} - \alpha_i)v_i^2$, $\theta_{i,i+1} = \theta_{i+1,i} = -1/(\alpha_{i+1} - \alpha_i)v_i v_{i+1}$ for $1 \le i \le n-1$, $\theta_{nn} = 1/(\alpha_n - \alpha_{n-1})v_n^2$, and $\theta_{ij} = 0$ otherwise.*

By Lemma 5 and Lemma 6, $\mathcal{ECWS}$ can be solved in $O(n)$ time. To solve $\mathcal{WS}$, in practice, the active set method takes less than n iterations and hence the total runtime is $O(n^2)$. Note that unlike previous works, the runtime of this convex quadratic programming approach is independent of the wire length L.

Applications

The wire sizing technique is commonly applied to minimize the wire delay and hence to improve the performance of VLSI circuits. As there are typically millions of wires in modern VLSI circuits, and each wire may be sized many many times in order to explore different architecture, logic design and layout during the design process, it is very important for wire sizing algorithms to be very efficient.

Another popular technique for delay minimization of slow signals is to insert buffers (or called repeaters) to strengthen and accelerate the signals. The work of Chu and Wong can be naturally extended to simultaneously handle buffer insertion. It is shown in [4] that the delay minimization problem for a wire by simultaneous buffer insertion and wire sizing can also be formulated as a convex quadratic program and be solved by active set method. The runtime is only m times more than that of wire sizing, where m is the number of buffers inserted. m is typically 5 or less in practice.

About one third of all nets in a typical VLSI circuit are multi-pin nets (i. e., nets with a tree structure to deliver a signal from a source to several sinks). It is important to minimize the delay of multi-pin nets. The work of Chu and Wong can also be applied to optimize multi-pin nets. The extension is described in Mo and Chu [14]. The idea is to integrate the quadratic programming approach into a dynamic programming framework. Each branch of the net is solved as a convex quadratic program while the overall tree structure is handled by dynamic programming.

Open Problems

After more than a decade of active research, the wire sizing problem by itself is now considered a well-solved problem. Some important solutions are [1,2,3,4,5,6,7,8,9,10,11,13,14,15]. The major remaining challenge is to simultaneously apply wire sizing with other interconnect optimization techniques to improve circuit performance. Wire sizing, buffer insertion and gate sizing are three most commonly used interconnect optimization techniques. It has been demonstrated that better performance can be achieved by simultaneously applying these three techniques than applying them sequentially. One very practical problem is to perform simultaneous wire sizing, buffer insertion and gate sizing to a combinational circuit such that the delay of all input-to-output paths are less than a given target and the total wire/buffer/gate resource usage is minimized.

Cross References

► Circuit Retiming
► Circuit Retiming: An Incremental Approach
► Gate Sizing

Recommended Reading

1. Chen, C.-P., Chen, Y.-P., Wong, D.F.: Optimal wire-sizing formula under the Elmore delay model. In: Proc. ACM/IEEE Design Automation Conf., pp. 487–490 ACM, New York (1996)
2. Chen, C.-P., Wong, D.F.: A fast algorithm for optimal wire-sizing under Elmore delay model. In: Proc. IEEE ISCAS, vol. 4, pp. 412–415 IEEE Press, Piscataway (1996)
3. Chen, C.-P., Wong, D.F.: Optimal wire-sizing function with fringing capacitance consideration. In: Proc. ACM/IEEE Design Automation Conf., pp. 604–607 ACM, New York (1997)
4. Chu, C.C.N., Wong, D.F.: Greedy wire-sizing is linear time. IEEE Trans. Comput. Des. **18**(4), 398–405 (1999)
5. Chu, C.C.N., Wong, D.F.: A quadratic programming approach to simultaneous buffer insertion/sizing and wire sizing. IEEE Trans. Comput. Des. **18**(6), 787–798 (1999)
6. Cong, J., He, L.: Optimal wiresizing for interconnects with multiple sources. ACM Trans. Des. Autom. Electron. Syst. **1**(4) 568–574 (1996)
7. Cong, J., Leung, K.-S.: Optimal wiresizing under the distributed Elmore delay model. IEEE Trans. Comput. Des. **14**(3), 321–336 (1995)
8. Fishburn., J.P.: Shaping a VLSI wire to minimize Elmore delay. In: Proc. European Design and Test Conference pp. 244–251. IEEE Compute Society, Washington D.C. (1997)
9. Fishburn, J.P., Schevon, C.A.: Shaping a distributed-RC line to minimize Elmore delay. IEEE Trans. Circuits Syst.-I: Fundam. Theory Appl. **42**(12), 1020–1022 (1995)
10. Gao, Y., Wong, D.F.: Wire-sizing for delay minimization and ringing control using transmission line model. In: Proc. Conf. on Design Automation and Test in Europe, pp. 512–516. ACM, New York (2000)
11. Kay, R., Bucheuv, G., Pileggi, L.: EWA: Efficient Wire-Sizing Algorithm. In: Proc. Intl. Symp. on Physical Design, pp. 178–185. ACM, New York (1997)
12. Kozlov, M.K., Tarasov, S.P., Khachiyan, L.G.: Polynomial solvability of convex quadratic programming. Sov. Math. Dokl. **20**, 1108–1111 (1979)
13. Lillis, J., Cheng, C.-K., Lin, T.-T.: Optimal and efficient buffer insertion and wire sizing. In: Proc. of Custom Integrated Circuits Conf., pp. 259–262. IEEE Press, Piscataway (1995)

14. Mo, Y.-Y., Chu, C.: A hybrid dynamic/quadratic programming algorithm for interconnect tree optimization. IEEE Trans. Comput. Des. **20**(5), 680–686 (2001)
15. Sapatnekar, S.S.: RC interconnect optimization under the Elmore delay model. In: Proc. ACM/IEEE Design Automation Conf., pp. 387–391. ACM, New York (1994)

Work-Function Algorithm for k Servers

1994; Koutsoupias, Papadimitriou

Marek Chrobak
Department of Computer Science at Riverside,
University of California at Riverside,
Riverside, CA, USA

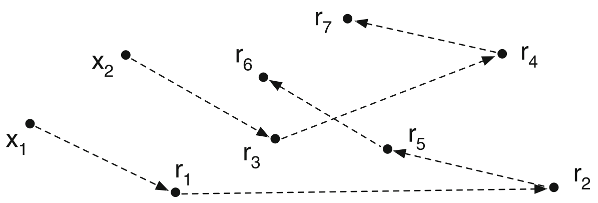
Problem Definition

In the *k-server problem*, the task is to schedule the movement of k servers in a metric space $\mathbb{M}$ in response to a sequence $\varrho = r_1, r_2, \dots, r_n$ of *requests*, where $r_i \in \mathbb{M}$ for all i. The servers initially occupy some configuration $X_0 \subseteq \mathbb{M}$. After each request r_i is issued, one of the k servers must move to r_i. A *schedule* S specifies which server moves to each request. The task is to compute a schedule with minimum *cost*, where the cost of a schedule is defined as the total distance traveled by the servers. The example below shows a schedule for 2 servers on a sequence of requests.

In the offline case, given $\mathbb{M}$, X_0, and the complete request sequence ϱ, the optimal schedule can be computed in polynomial time [6].

In the *online* version of the problem the decision as to which server to move to each request r_i must be made before the next request r_{i+1} is issued. It is quite easy to see that in this online scenario it is not possible to guarantee an optimal schedule. The accuracy of online algorithms is often measured using competitive analysis. Denote by $cost_{\mathcal{A}}(\varrho)$ the cost of the schedule produced by an online k-server algorithm $\mathcal{A}$ on a request sequence ϱ, and let $opt(\varrho)$ be the cost of an optimal schedule on ϱ. $\mathcal{A}$ is called *R-competitive* if $cost_{\mathcal{A}}(\varrho) \le R \cdot opt(\varrho) + B$, where B is a constant that may depend on $\mathbb{M}$ and X_0. The smallest such R is called the *competitive ratio* of $\mathcal{A}$. Of course, the smaller the R the better.

The k-server problem was introduced by Manasse, McGeoch, and Sleator [13,14], who proved that no (deterministic) on-line algorithm can achieve a competitive ratio smaller than k, in any metric space with at least $k+1$ points. They also gave a 2-competitive algorithm for $k = 2$ and stated what is now known as the *k-server conjecture*,



Work-Function Algorithm for k Servers, Figure 1
A schedule for 2 servers on a request sequence $\varrho = r_1, r_2, \dots, r_7$. The initial configuration is $X_0 = \{x_1, x_2\}$. Server 1 serves r_1, r_2, r_5, r_6, while server 2 serves r_3, r_4, r_7. The cost of this schedule is $d(x_1,r_1) + d(r_1,r_2) + d(r_2,r_5) + d(r_5,r_6) + d(x_2,r_3) + d(r_3,r_4) + d(r_4,r_7)$, where $d(x,y)$ denotes the distance between points x, y

which postulates that there exists a k-competitive online algorithm for all k. Koutsoupias and Papadimitriou [10,11] (see also [3,8,9]) proved that the *work-function algorithm* presented in the next section has competitive ratio at most $2k-1$, which to date remains the best upper bound known.

Key Results

The idea of the work-function algorithm is to balance two greedy strategies when a new request is issued. The first one is to simply serve the request with the closest server. The second strategy attempts to follow the optimum schedule. Roughly, from among the k possible new configurations, this strategy chooses the one where the optimum schedule would be at this time, if no more requests remained to be issued.

To formalize this idea, for each request sequence ϱ and a k-server configuration X, let $\omega_\varrho(X)$ be the minimum cost of serving ϱ under the constraint that at the end the server configuration is X. (Assume that the initial configuration X_0 is fixed.) The function $\omega_\varrho(\cdot)$ is called the *work function* after the request sequence ϱ.

Algorithm WFA

Denote by σ the sequence of past requests, and suppose that the current server configuration is $S = \{s_1, s_2, \dots, s_k\}$, where s_j is the location of the jth server. Let r be the new request. Choose $s_j \in S$ that minimizes the quantity $\omega_{\sigma r}(S - \{s_j\} \cup \{r\}) + d(s_j, r)$, and move server j to r.

Theorem 1 ([10,11]) *Algorithm* WFA *is* $(2k-1)$*-competitive.*

Applications

The k-server problem can be viewed as an abstraction of online problems that arise in emergency crew scheduling, caching (or paging) in two-level memory systems, scheduling of disk heads, and other. Nevertheless, in its pure abstract form, it is mostly of theoretical interest.

Algorithm WFA can be applied to some generalizations of the k-server problem. In particular, it is $(2n - 1)$-competitive for n-state metrical task systems, matching the lower bound [3,4,8]. See [1,3,5] for other applications and extensions.

Open Problems

Theorem 1 comes tantalizingly close to settling the k-server conjecture described earlier in this section. In fact, it has been even conjectured that Algorithm WFA itself is k-competitive for k servers, but the proof of this conjecture, so far, remains elusive.

For $k \geq 3$, k-competitive online k-server algorithms are known only for some restricted metric spaces, including trees [7], metric spaces with up to $k + 2$ points, and the Manhattan plane for $k = 3$ (see [2,6,12]). As the analysis of Algorithm WFA in the general case appears difficult, it would of interest to prove its k-competitiveness for some natural special cases, for example in the plane (with any reasonable metric) for $k \geq 4$ servers.

Very little is known about the competitive ratio of the k-server problem in the randomized case. In fact, it is not even known whether a ratio better than 2 can be achieved for $k = 2$.

Cross References

▶ Algorithm DC-Tree for k Servers on Trees
▶ Deterministic Searching on the Line
▶ Generalized Two-Server Problem
▶ Metrical Task Systems
▶ Online Paging and Caching
▶ Paging

Recommended Reading

1. Anderson, E.J., Hildrum, K., Karlin, A.R., Rasala, A., Saks, M.: On list update and work function algorithms. Theor. Comput. Sci. **287**, 393–418 (2002)
2. Bein, W., Chrobak, M., Larmore, L.L.: The 3-server problem in the plane. Theor. Comput. Sci. **287**, 387–391 (2002)
3. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press, Cambridge (1998)
4. Borodin, A., Linial, N., Saks, M.: An optimal online algorithm for metrical task systems. In: Proc. 19th Symp. Theory of Computing (STOC), ACM, pp. 373–382 (1987)
5. Burley, W.R.: Traversing layered graphs using the work function algorithm. J. Algorithms **20**, 479–511 (1996)
6. Chrobak, M., Karloff, H., Payne, T.H., Vishwanathan, S.: New results on server problems. SIAM J. Discret. Math. **4**, 172–181 (1991)
7. Chrobak, M., Larmore, L.L.: An optimal online algorithm for k servers on trees. SIAM J. Comput. **20**, 144–148 (1991)
8. Chrobak, M., Larmore, L.L.: Metrical task systems, the server problem, and the work function algorithm. In: Fiat, A., Woeginger, G.J. (eds.) Online Algorithms: The State of the Art, pp. 74–94. Springer, London (1998)
9. Koutsoupias, E.: Weak adversaries for the k-server problem. In: Proc. 40th Symp. Foundations of Computer Science (FOCS), IEEE, pp. 444–449 (1999)
10. Koutsoupias, E., Papadimitriou, C.: On the k-server conjecture. In: Proc. 26th Symp. Theory of Computing (STOC), ACM, pp. 507–511 (1994)
11. Koutsoupias, E., Papadimitriou, C.: On the k-server conjecture. J. ACM **42**, 971–983 (1995)
12. Koutsoupias, E., Papadimitriou, C.: The 2-evader problem. Inf. Proc. Lett. **57**, 249–252 (1996)
13. Manasse, M., McGeoch, L.A., Sleator, D.: Competitive algorithms for online problems. In: Proc. 20th Symp. Theory of Computing (STOC), ACM, pp. 322–333 (1988)
14. Manasse, M., McGeoch, L.A., Sleator, D.: Competitive algorithms for server problems. J. Algorithms **11**, 208–230 (1990)

X

XML Compression and Indexing

▶ Tree Compression and Indexing

Chronological Index

Bibliography

Aardal, K., Chudak, F.A., Shmoys, D.B.: A 3-approximation algorithm for the *k*-level uncapacitated facility location problem. Inf. Process. Lett. **72**, 161–167 (1999)

Aaronson, S., Ambainis, A.: Quantum search of spatial regions. In: Proc. 44th Annual IEEE Symp. on Foundations of Computer Science (FOCS), 2003, pp. 200–209

Aaronson, S., Ambainis A.: Quantum search of spatial regions. Theor. Comput. **1**, 47–79 (2005)

Aaronson, S., Shi, Y.: Quantum Lower Bounds for the Collision and the Element Distinctness Problems. J. ACM **51**(4), 595–605 (2004)

Abam, M.A., de Berg, M., Farshi, M., Gudmundsson, J.: Region-fault tolerant geometric spanners. In: Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms, New Orleans, 7–9 January 2007

Abdi, H.: Additive-tree representations. In: Dress, A., von Haeseler, A. (eds.) Trees and Hierarchical Structures: Proceedings of a conference held at Bielefeld, FRG, Oct. 5–9th, 1987. Lecture Notes in Biomathematics, vol. 84, pp. 43–59. Springer (1990)

Abdulkadiroğlu, A., Pathak, P.A., Roth, A.E.: The New York City high school match. Am. Economic. Rev. **95**(2), 364–367 (2006)

Abdulkadiroĝlu, A., Sönmez, T.: Random serial dictatorship and the core from random endowments in house allocation problems. Econom. **66**(3), 689–701 (1998)

Abeysekera, S.P., Turtle H.J.: Long-run relations in exchange markets: a test of covered interest parity. J. Financial Res. **18**(4), 431–447 (1995)

Abouelhoda, M.I., Kurtz, S., Ohlebusch, E.: Replacing suffix trees with enhanced suffix arrays. J. Discret. Algorithms **2**, 53–86 (2004)

Abraham, D., Blum, A., Sandholm, T.: Clearing algorithms for barter exchange markets: Enabling nationwide kidney exchanges. EC'07, June 11–15, 2007, San Diego, California

Abraham, D., Cechlárová, K., Manlove, D., Mehlhorn, K.: Pareto-optimality in house allocation problems. In: Fleischer, R., Trippen, G. (eds.) Lecture Notes in Comp. Sci. Vol. 3341/2004, Algorithms and Computation, 14th Int. Symposium ISAAC 2004, pp. 3–15. Hong Kong, December 2004

Abraham, D.J., Cechlárová, K., Manlove, D.F., Mehlhorn, K.: Pareto-optimality in house al- location problems. In: Proceedings of the 15th International Symposium on Algorithms and Computation, (LNCS, vol. 3341), pp. 3–15. Springer, Sanya, Hainan (2004)

Abraham, D.J., Chen, N., Kumar, V., Mirrokni, V.: Assignment problems in rental markets. In: Proceedings of the 2nd Workshop on Internet and Network Economics, Patras, December 15–17 2006

Abraham, D.J., Irving, R.W., Kavitha, T., Mehlhorn, K.: Popular matchings. In: Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 424–432 (2005)

Abraham, D.J., Irving, R.W., Manlove, D.F.: Two algorithms for the Student-Project allocation problem. J. Discret. Algorithms **5**(1), 73–90 (2007)

Abraham, D.J., Kavitha, T.: Dynamic matching markets and voting paths. In: Proceedings of the 10th Scandinavian Workshop on Algorithm Theory (SWAT), pp. 65–76, Riga, July 6–8 2006

Abraham, I., Awerbuch, B., Azar, Y., Bartal, Y., Malkhi, D., Pavlov, E.: A generic scheme for building overlay networks in adversarial scenarios. In: Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS 2003), 2003

Abraham, I., Badola, A., Bickson, D., Malkhi, D., Maloo, S., Ron, S.: Practical locality-awareness for large scale information sharing. In: The 4th Annual International Workshop on Peer-To-Peer Systems (IPTPS '05), 2005

Abraham, I., Bartal, Y., Neiman, O.: Embedding Metrics into Ultrametrics and Graphs into Spanning Trees with Constant Average Distortion. In: Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms, New Orleans, January 2007

Abraham, I., Malkhi, D., Dobzinski, O.: LAND: Stretch $(1 + \varepsilon)$ locality aware networks for DHTs. In: Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA04), 2004

Abrams, Z.: Revenue maximization when bidders have budgets. In: Proceedings of the 17th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA-06), Miami, FL 2006, pp. 1074–1082, ACM Press, New York (2006)

Abu-Khzam, F., Collins, R., Fellows, M., Langston, M., Suters, W., Symons, C.: Kernelization algorithms for the vertex cover problem: theory and experiments. In: Proc. Workshop on Algorithm Engineering and Experiments (ALENEX) pp. 62–69 (2004)

Acar, U.A., Blelloch, G.E., Harper, R., Vittes, J.L., Woo, S.L.M.: Dynamizing static algorithms, with applications to dynamic trees and history independence. In: Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 524–533. SIAM (2004)

Acar, U.A., Blelloch, G.E., Vittes, J.L.: An experimental analysis of change propagation in dynamic trees. In: Proceedings of the 7th Workshop on Algorithm Engineering and Experiments (ALENEX), pp. 41–54 (2005)

Achioptas, D., Sorkin, G.B.: Optimal myopic algorithms for random 3-sat. In: 41st Annual Symposium on Foundations of Computer Science, pp. 590–600. IEEE Computer Society, Washington (2000)

Achlioptas, D.: Lower bounds for random 3-sat via differential equations. Theor. Comput. Sci. **265**(1–2), 159–185 (2001)

Ackermann, H., Goldberg, P., Mirrokni, V., Röglin, H., Vöcking, B.: A unified Approach to Congestion Games and Two-sided markets. In: 3rd Workshop of Internet Economics (WINE), pp. 30–41. San Diego, CA, USA (2007)

Ackermann, H., Goldberg, P., Mirrokni, V., Röglin, H., Vöcking, B.: Uncoordinated two-sided markets. ACM Electronic Commerce (ACM EC) (2008)

ACM Journal of Experimental Algorithmics. Launched in 1996, this journal publishes contributed articles as well as special sections containing selected papers from ALENEX and WEA. Visit www.jea.acm.org, or visit portal.acm.org and click on ACM Digital Library/Journals/Journal of Experimental Algorithmics

Adamy, U., Erlebach, T.: Online coloring of intervals with bandwidth. In: Proc. of the First International Workshop on Approximation and Online Algorithms (WAOA2003), pp. 1–12 (2003)

Adler, M., Khanna, S., Rajaraman, R., Rosén, A.: Time-constrained scheduling of weighted packets on trees and meshes. Algorithmica **36**, 123–152 (2003)

Afek, Y., Attiya, H., Dolev, D., Gafni, E., Merritt, M., Shavit, N.: Atomic snapshots of shared memory. J. Assoc. Comput. Mach. **40**, 873–890 (1993)

Afek, Y., Mansour, Y., Ostfeld, Z.: Convergence complexity of optimistic rate based flow control algorithms. J. Algorithms **30**(1), 106–143 (1999)

Afek, Y., Mansour, Y., Ostfeld, Z.: Phantom: a simple and effective flow control scheme. Comput. Netw. **32**(3), 277–305 (2000)

Afek, Y., Stupp, G., Touitou, D.: Long lived adaptive splitter and applications. Distrib. Comput. **30**, 67–86 (2002)

Afrati, F.N., Bampis, E., Chekuri, C., Karger, D.R., Kenyon, C., Khanna, S., Milis, I., Queyranne, M., Skutella, M., Stein, C., Sviridenko, M.: Approximation schemes for minimizing average weighted completion time with release dates. In: Proc. of Foundations of Computer Science, pp. 32–44 (1999)

Agarwal, A., Charikar, M., Makarychev, K., Makarychev, Y.: $O(\sqrt{\log n})$ approximation algorithms for Min UnCut, Min 2CNF Deletion, and directed cut problems. In: Proceedings of the 37th ACM Symposium on Theory of Computing (STOC), pp. 573–581, Baltimore, May 2005

Agarwal, A., Cherian, M.: Adaptive backoff synchronization techniques. In: Proceedings of the 16th Annual International Symposium on Computer Architecture, pp. 396–406. ACM Press, New York (1989)

Agarwal, P.K., Arge, L., Danner, A., Holland-Minkley, B.: Cache-oblivious data structures for orthogonal range searching. In: Proc. 19th ACM Symposium on Computational Geometry, pp. 237–245. ACM, New York (2003)

Agarwal, P.K., de Berg, M., Gudmundsson, J., Hammar, M., Haverkort, H.J.: Box-trees and R-trees with near-optimal query time. Discret. Comput. Geom. **28**, 291–312 (2002)

Agarwal, P.K., Edelsbrunner, H., Schwarzkopf, O., Welzl, E.: Euclidean minimum spanning trees and bichromatic closest pairs. Discret. Comput. Geom. **6**, 407–422 (1991)

Agarwal, P.K., Flato, E., Halperin, D.: Polygon decomposition for efficient construction of Minkowski sums. Comput. Geom. Theor. Appl. **21**(1–2), 39–61 (2002)

Agarwal, P.K., Har-Peled, S., Karia, M.: Computing approximate shortest paths on convex polytopes. In: Proceedings of the 16th ACM Symposium on Computational Geometry, pp. 270–279. ACM Press, New York (2000)

Agarwala, R., Fernández-Baca, D.: A polynomial-time algorithm for the perfect phylogeny problem when the number of character states is fixed. SIAM J. Comput. **23**, 1216–1224 (1994)

Agarwala, R., Fernández-Baca, D., Slutzki, G.: Fast algorithms for inferring evolutionary trees. J. Comput. Biol. **2**, 397–407 (1995)

Ageev, A.A., Sviridenko, M.I.: An 0.828-approximation algorithm for the uncapacitated facility location problem. Discret. Appl. Math. **93**, 149–156 (1999)

Aggarwal, A., Alon, N., Charikar, M.: Improved approximations for directed cut problems. In: Proceedings of the 39th ACM Symposium on Theory of Computing (STOC), pp. 671–680, San Diego, June 2007

Aggarwal, A., Plaxton, C.G.: Optimal parallel sorting in multi-level storage. In: Proceedings of the ACM-SIAM Symposium on Discrete Algorithms, vol. 5, pp. 659–668. ACM Press, New York (1994)

Aggarwal, A., Vitter, J.S.: The input/output complexity of sorting and related problems. Commun. ACM **31**(9), 1116–1127 (1988)

Aggarwal, G., Fiat, A., Goldberg, A., Immorlica, N., Sudan, M.: Derandomization of auctions. In: Proc. of the 37th ACM Symposium on Theory of Computing (STOC'05), 2005

Aggarwal, G., Muthukrishnan, S., Feldman, J.: Bidding to the top: Vcg and equilibria of position-based auctions. http://www.citebase.org/abstract?id=oai:arXiv.org:cs/0607117 (2006)

Agmon., S.: The relaxation method for linear inequalities. Can. J. Math. **6**(3), 382–392 (1954)

Agnarsson, G., Halldórsson, M.M.: Coloring Powers of Planar Graphs. In: Proceedings of the 11th Annual ACM-SIAM symposium on Discrete algorithms, pp. 654–662 (2000)

Agrawal, A., Klein, P., Ravi, R.: When trees collide: An approximation algorithm for the generalized Steiner problem in networks. SIAM J. Comput. **24**(3), 445–456 (1995)

Agrawal, A., Klein, P., Ravi, R.: When trees collide: an approximation algorithm for the generalized Steiner problem on networks. In: Proc. of the 23rd Annual ACM Symposium on Theory of Computing, Association for Computing Machinery, New York, pp. 134–144 (1991)

Agrawal, A., Klein, P.N., Ravi, R.: Cutting down on fill using nested dissection: provably good elimination orderings. In: Brualdi, R.A., Friedland, S., Klee, V. (eds.) Graph theory and sparse matrix computation. IMA Volumes in mathematics and its applications, pp. 31–55. Springer, New York (1993)

Aguilera, M.K., Delporte-Gallet, C., Fauconnier, H., Toueg, S.: On implementing Omega with weak reliability and synchrony assumptions. In: 22th ACM Symposium on Principles of Distributed Computing, pp. 306–314 (2003)

Aguilera, M.K., Frolund, S., Hadzilacos, V., Horn, S.L., Toueg, S.: Brief announcement: Abortable and query-abortable objects. In: Proc. 20th Annual International Symposium on Distributed Computing, 2006

Aharonov, D., Ambainis, A., Kempe, J., Vazirani, U.: Quantum walks on graphs. In: Proc. STOC (2001)

Aharonov, D., Arad, I.: The BQP-hardness of approximating the Jones Polynomial. arxiv: quant-ph/0605181 (2006)

Aharonov, D., Arad, I., Eban, E., Landau, Z.: Polynomial Quantum Algorithms for Additive approximations of the Potts model and other Points of the Tutte Plane. arxiv:quant-ph/0702008 (2007)

Aharonov, D., Ben-Or, M.: Fault-tolerant quantum computation with constant error rate. In: Proc. 29th ACM Symp. on Theory of Computing (STOC), pp. 176–188, (1997). quant-ph/9906129

Aharonov, D., Jones, V., Landau, Z.: A polynomial quantum algorithm for approximating the Jones polynomial. Proceedings of the 38th ACM Symposium on Theory of Computing (STOC) Seattle, Washington, USA, arxiv:quant-ph/0511096 (2006)

Aharonov, D., Kitaev, A.Y., Preskill, J.: Fault-tolerant quantum computation with long-range correlated noise. Phys. Rev. Lett. **96**, 050504 (2006). quant-ph/0510231

Ahmed, N., Kanhere, S.S., Jha, S.: The holes problem in wireless sensor networks: a survey. SIGMOBILE Mob. Comput. Commun. Rev. **9**, 4–18 (2005)

Aho, A.: Algorithms for Finding Patterns in Strings. In: van Leewen, J. (ed.) Handbook of Theoretical Computer Science, vol. A: Algorithms and Complexity, pp. 255–300. Elsevier Science, Amsterdam and MIT Press, Cambridge (1990)

Aho, A., Johnson, S.: Optimal Code Generation for Expression Trees. J. ACM **23**(July), 488–501 (1976)

Aho, A., Sethi, R., Ullman, J.: Compilers: Principles, Techniques and Tools. pp. 557–584. Addison Wesley, Boston (1986)

Aho, A.V., Corasick, M.J.: Efficient string matching: an aid to bibliographic search. C. ACM **18**(6), 333–340 (1975)

Aho, A.V., Hopcroft, J.E., Ullman, J.D.: The design and analysis of computer algorithms. Addison-Wesley, Reading (1975)

Aho, A.V., Hopcroft, J.E., Ullman, J.D.: The Design and Analysis of Computer Algorithms. Addison-Wesley (1974)

Ahrens, J.H., Dieter, U.: Sequential random sampling. ACM Trans. Math. Softw. **11**, 157–169 (1985)

Ahuja, R., Magnanti, T., Orlin, J.: Network Flows. Prentice-Hall, Englewood Cliffs (1993)

Ahuja, R.K., Magnanti, T.L., Orlin, J.B., Reddy, M.R.: Applications of network optimization. In: Handbooks in Operations Research and Management Science, vol. 7, Network Models, chapter 1, pp. 1–83. North-Holland, Amsterdam (1995)

Ahuja, R.K., Magnati, T.L., Orlin, J.B.: Network Flows: Theory, Algorithms, and Applications. Prentice Hall, Englewood Cliffs (1993)

Aiello, W., Mansour, Y., Rajagopolan, S., Rosen, A.: Competitive queue policies for differentiated services. In: Proc. of the IEEE INFOCOM, pp. 431–440. IEEE, Tel-Aviv, Israel (2000)

Aingworth, D., Chekuri, C., Indyk, P., Motwani, R.: Fast estimation of diameter and shortest paths (without matrix multiplication). SIAM J. Comput. **28**, 1167–1181 (1999)

Aingworth, D., Chekuri, C., Motwani, R.: Fast estimation of diameter and shortest paths (without matrix multiplication). In: Proc. 7th ACM-SIAM Symposium on Discrete Algorithms, 1996, pp. 547–553

Ajana, Y., Lefebvre, J.-F., Tillier, E., El-Mabrouk, N.: Exploring the Set of All Minimal Sequences of Reversals – An Application to Test the Replication-Directed Reversal Hypothesis, Proceedings of the Second Workshop on Algorithms in Bioinformatics. Lecture Notes in Computer Science, vol. 2452, pp. 300–315. Springer, Berlin (2002)

Ajtai, M.: A lower bound for finding predecessors in Yao's cell probe model. Combinatorica **8**(3), 235–247 (1988)

Ajtai, M.: $\sum_1^1$-formulae on finite structures. Ann. Pure Appl. Log. **24**(1), 1–48 (1983)

Ajtai, M., Kumar, R., Sivakumar, D.: A sieve algorithm for the shortest lattice vector problem. In: Proceedings of the thirty-third annual ACM symposium on theory of computing – STOC 2001, Heraklion, Crete, Greece, July 2001, pp 266–275. ACM, New York (2001)

Ajwani, D., Dementiev, U., Meyer, R., Osipov, V.: Breadth first search on massive graphs. In: 9th DIMACS Implementation Challenge Workshop: Shortest Paths, DIMACS Center, Piscataway, NJ, 13–14 Nov 2006

Akavia, A., Goldwasser, S.: Manuscript submitted as an NSF grant, awarded (2005) CCF-0514167

Akavia, A., Goldwasser, S., Safra, S.: Proving hard-core predicates using list decoding. In: Proceedings of the 44th Symposium on Foundations of Computer Science (FOCS'03), pp. 146–157. IEEE Computer Society (2003)

Akutsu, T.: Dynamic programming algorithms for RNA secondary structure prediction with pseudoknots. Discret. Appl. Math. **104**, 45–62 (2000)

Akutsu, T., Kanaya, K., Ohyama, A., Fujiyama, A.: Point matching under non-uniform distortions. Discret. Appl. Math. **127**, 5–21 (2003)

Akyildiz, I.F., Su, W., Sankarasubramaniam, Y., Cayirci, E.: Wireless sensor networks: a survey. J. Comput. Netw. **38**, 393–422 (2002)

Al-Karaki, J.N., Kamal, A.E.: Routing techniques in wireless sensor networks: a survey. Wirel. Commun. IEEE **11**, 6–28 (2004)

Aland, S., Dumrauf, D., Gairing, M., Monien, B., Schoppmann, F.: Exact price of anarchy for polynomial congestion games. In: 23rd Annual Symposium on Theoretical Aspects of Computer Science (STACS), pp. 218–229. Springer, Marseille (2006)

Alber, J., Betzler, N., Niedermeier, R.: Experiments on data reduction for optimal domination in networks. Ann. Oper. Res. **146**(1), 105–117 (2006)

Alber, J., Bodlaender, H.L., Fernau, H., Kloks, T., Niedermeier, R.: Fixed parameter algorithms for Dominating Set and related problems on planar graphs. Algorithmica **33**(4), 461–493 (2002)

Alber, J., Dorn, B., Niedermeier, R.: A general data reduction scheme for domination in graphs. In: Proc. 32nd SOFSEM. LNCS, vol. 3831, pp. 137–147. Springer, Berlin (2006)

Alber, J., Fan, H., Fellows, M.R., Fernau, H., Niedermeier, R., Rosamond, F., Stege, U.: A refined search tree technique for Dominating Set on planar graphs. J. Comput. Syst. Sci. **71**(4), 385–405 (2005)

Alber, J., Fellows, M.R., Niedermeier, R.: Polynomial time data reduction for Dominating Set. J. ACM **51**(3), 363–384 (2004)

Albers, S.: Better bounds for online scheduling. SIAM J. Comput. **29**(2), 459–473 (1999)

Albers, S.: Improved randomized on-line algorithms for the list update problem. SIAM J. Comput. **27**, 670–681 (1998)

Albers, S., Fujiwara, H.: Energy-efficient algorithms for flow time minimization. In: STACS. Lecture Notes in Computer Science, vol. 3884, pp. 621–633. Springer, Berlin (2006)

Albers, S., Henzinger, M.R.: Exploring unknown environments. SIAM J. Comput. **29**, 1164–1188 (2000)

Albers, S., Kursawe, K., Schuierer, S.: Exploring unknown environments with obstacles. Algorithmica **32**(1), 123–143 (2002)

Albers, S., Schmidt, M.: On the performance of greedy algorithms in packet buffering. SIAM J. Comput. **35**, 278–304 (2005)

Albers, S., von Stengel, B., Werchner, R.: A combined BIT and TIMESTAMP algorithm for the list update problem. Inf. Proc. Lett. **56**, 135–139 (1995)

Alberts, D., Cattaneo, G., Italiano, G.F.: An empirical study of dynamic graph algorithms. ACM J. Exp. Algorithmics **2** (1997)

Aldous, D., Fill, J.: Reversible markov chains and random walks on graphs. http://stat-www.berkeley.edu/users/aldous/book.html (1999). Accessed 1999

Alekhnovich, M., Braverman, M., Feldman, V., Klivans, A.R., Pitassi, T.: Learnability and automatizability. In: FOCS '04 Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS'04), pp. 621–630. IEEE Computer Society, Washington (2004)

ALENEX. Beginning in 1999, the annual workshop on Algorithm Engineering and Experimentation is sponsored by SIAM and ACM. It is co-located with SODA, the SIAM Symposium on Data Structures and Algorithms. Workshop proceedings are published in the Springer LNCS series. Visit www.siam.org/meetings/ for more information

Algorithmic Solutions Software GmbH, http://www.algorithmic-solutions.com/. Accessed February 2008

Alicherry, M., Bhatia, R., Li, L.E.: Joint channel assignment and routing for throughput optimization in multi-radio wireless mesh networks. In: Proc. ACM MOBICOM 2005, pp. 58–72

Aliferis, P., Gottesman, D., Preskill, J.: Quantum accuracy threshold for concatenated distance-3 codes. Quant. Inf. Comput. **6**, 97–165 (2006). quant-ph/0504218

Allauzen, C., Crochemore, M., Raffinot, M.: Factor oracle: a new structure for pattern matching. In: SOFSEM'99. LNCS, vol. 1725, pp. 291–306. Springer, Berlin (1999)

Allender, E., Arora, S., Kearns, M., Moore, C., Russell, A.: Note on the representational incompatabilty of function approximation and factored dynamics. In: Advances in Neural Information Processing Systems 15, 2002

Allgower, E.L., Schmidt, P.H.: An Algorithm for Piecewise-Linear Approximation of an Implicitly Defined Manifold. SIAM J. Num. Anal. **22**, 322–346 (1985)

Alon, N., Asodi, V.: Learning a hidden subgraph, ICALP. LNCS **3142**, 110–121 (2004). Also: SIAM J. Discret. Math. **18**, 697–712 (2005)

Alon, N., Awerbuch, B., Azar, Y., Buchbinder, N., Naor, J.: A general approach to online network optimization problems. In: Symposium on Discrete Algorithms, pp. 570–579 (2004)

Alon, N., Bar-Noy, A., Linial, N., Peleg, D.: A lower bound for radio broadcast. J. Comput. Syst. Sci. **43**(2), 290–298 (1991)

Alon, N., Beigel, R., Kasif, S., Rudich, S., Sudakov, B.: Learning a Hidden Matching, Proceedings of the 43rd IEEE FOCS, 2002, 197–206. Also: SIAM J. Computing **33**, 487–501 (2004)

Alon, N., Chung, F., Graham, R.: Routing permutations on graphs via matching. SIAM J. Discret. Math. **7**(3), 513–530 (1994)

Alon, N., Galil, Z., Margalit, O.: On the exponent of the all pairs shortest path problem. In: Proc. 32th IEEE FOCS, pp. 569–575. IEEE Computer Society, Los Alamitos, USA (1991). Also JCSS **54**, 255–262 (1997)

Alon, N., Galil, Z., Margalit, O., Naor, M.: Witnesses for Boolean matrix multiplication and for shortest paths. In: Proc. 33th IEEE FOCS, pp. 417–426. IEEE Computer Society, Los Alamitos, USA (1992)

Alon, N., Goldreich, O., Håstad, J., Peralta, R.: Simple constructions of almost *k*-wise independent random variables. Random Struct. Algorithms **3**(3), 289–304 (1992)

Alon, N., Kahale, N.: A spectral technique for coloring random 3-colorable graphs. SIAM J. Comput. **26**(6), 1733–1748 (1997)

Alon, N., Karp, R.M., Peleg, D., West, D.: A graph-theoretic game and its application to the *k*-server problem. SIAM J. Comput. **24**, 78–100 (1995)

Alon, N., Kaufman, T., Krivilevich, M., Litsyn, S., Ron, D.: Testing low-degree polynomials over gf(2). In: Proceedings of RANDOM '03. Lecture Notes in Computer Science, vol. 2764, pp. 188–199. Springer, Berlin Heidelberg (2003)

Alon, N., Spencer, J.: The Probabilistic Method. Wiley (1992)

Alon, N., Spencer, J.H.: The Probabilistic Method. 2nd edn. Wiley, New York (2000)

Alon, N., Spencer, J.H.: The Probabilistic Method. Wiley, New York (1991)

Alon, N., Yuster, R., Zwick, U.: Color coding. J. ACM **42**, 844–856 (1995)

Alon, N., Yuster, R., Zwick, U.: Finding and counting given length cycles. Algorithmica **17**(3), 209–223 (1997)

Alpern, S., Gal, S.: The Theory of Search Games and Rendezvous. Kluwer Academic Publishers, Norwell (2003)

Alpert, C.J., Chan, T., Kahng, A.B., Markov, I.L., Mulet, P.: Faster minimization of linear wirelength for global placement. IEEE Trans. CAD **17**(1), 3–13 (1998)

Alpert, C.J., Kahng, A.B.: Recent directions in netlist partitioning: a survey. Integr. VLSI J. **19**(1–2), 1–81 (1995)

Alstrup, S., Brodal, G.S., Rauhe, T.: Pattern matching in dynamic texts. In: Proc. of Symposium on Discrete Algorithms (SODA), 2000, pp. 819–828

Alstrup, S., Holm, J., de Lichtenberg, K., Thorup, M.: Direct routing on trees. In: Proceedings of the Ninth Annual ACM-SIAM, Symposium on Discrete Algorithms (SODA 98), pp. 342–349. San Francisco, California, United States (1998)

Alstrup, S., Holm, J., de Lichtenberg, K., Thorup, M.: Minimizing diameters of dynamic trees. In: Proceedings of the 24th International Colloquium on Automata, Languages and Programming (ICALP), Bologna, Italy, 7–11 July 1997. Lecture Notes in Computer Science, vol. 1256, pp. 270–280. Springer (1997)

Alstrup, S., Holm, J., Thorup, M., de Lichtenberg, K.: Maintaining information in fully dynamic trees with top trees. ACM Trans. Algorithms **1**(2), 243–264 (2005)

Alstrup, S., Husfeldt, T., Rauhe, T.: Marked ancestor problems. In: Proc. 39th IEEE Symposium on Foundations of Computer Science (FOCS), 1998, pp. 534–543

Alstrup, S., Husfeldt, T., Rauhe, T., Thorup, M.: Black box for constant-time insertion in priority queues (note). ACM TALG **1**(1), 102–106 (2005)

Alt, H., Guibas, L.: Discrete geometric shapes: Matching, interpolation, and approximation. In: Sack, J.R., Urrutia, J. (eds.) Handbook of Computational Geometry, pp. 121–153. Elsevier Science Publishers B.V. North-Holland, Amsterdam (1999)

Alt, H., Mehlhorn, K., Wagener, H., Welzl, E.: Congruence, similarity and symmetries of geometric objects. Discret. Comput. Geom. **3**, 237–256 (1988)

Althaus, E., Mehlhorn, K.: Traveling salesman-based curve reconstruction in polynomial time. SIAM J. Comput. **31**, 27–66 (2001)

Althaus, E., Mehlhorn, K., Näher, S., Schirra, S.: Experiments on curve reconstruction. In: ALENEX, 2000, pp. 103–114

Althöfer, I.: On sparse approximations to randomized strategies and convex combinations. Linear Algebr. Appl. **199**, 339–355 (1994)

Althofer, I., Das, G., Dobkin, D.P., Joseph, D., Soares, J.: On Sparse Spanners of Weighted Graphs. Discret. Comput. Geom. **9**, 81–100 (1993)

Altinel, M., Franklin, M.: Efficient filtering of XML documents for selective dissemination of information. In: *Proceedings of 26th International Conference on Very Large Data Bases*, Cairo, Egypt, pp. 53–64. Morgan Kaufmann, Missouri (2000)

Altschul, S.F., Erickson, B.W.: Optimal sequence alignment using affine gap costs. Bull. Math. Biol. **48**, 603–616 (1986)

Altschul, S.F., Gish, W., Miller, W., Myers, E.W., Lipman, D.J.: Basic Local Alignment Search Tool. J. Mol. Biol. **215**, 403–410 (1990)

Alur, R., Taubenfeld, G.: Results about fast mutual exclusion. In: Proceedings of the 13th IEEE Real-Time Systems Symposium, December 1992, pp. 12–21

Aluru, S. (ed.): Handbook of Computational Molecular Biology. Computer and Information Science Series. Chapman and Hall/CRC Press, Boca Raton (2005)

Alzoubi, K., Li, X.-Y., Wang, Y., Wan, P.-J., Frieder, O.: Geometric spanners for wireless ad hoc networks. IEEE Trans. Parallel Distrib. Process. **14**, 408–421 (2003)

Alzoubi, K., Wan, P.-J., Frieder, O.: New distributed algorithm for connected dominating set in wireless ad hoc networks. In: Proceedings of IEEE 35th Hawaii International Conference on System Sciences (HICSS-35), Hawaii, 7–10 January 2002

Alzoubi, K.M., Wan, P.-J., Frieder, O.: Message-optimal connected dominating sets in mobile ad hoc networks. In: ACM MOBIHOC, Lausanne, Switzerland, 09–11 June 2002

Amato, N.M., Goodrich, M.T., Ramos, E.A.: Computing the arrangement of curve segments: Divide-and-conquer algorithms via sampling. In: Proc. 11th Annual ACM-SIAM Symp. on Discrete Algorithms, pp. 705–706 (2000)

Ambainis, A.: A nearly optimal discrete query quantum algorithm for evaluating NAND formulas, arXiv:0704.3628 (2007)

Ambainis, A.: Polynomial degree and lower bounds in quantum complexity: Collision and element distinctness with small range. Theor. Comput. **1**, 37–46 (2005)

Ambainis, A.: Quantum lower bounds by quantum arguments. J. Comput. Syst. Sci. **64**, 750–767, (2002), quant-ph/0002066

Ambainis, A.: Quantum walk algorithm for Element Distinctness. In: Proceedings of the 45th Symposium on Foundations of Computer Science, pp. 22–31, Rome, Italy, 17–19 October 2004

Ambainis, A.: Quantum walk algorithm for element distinctness. SIAM J. Comput. **37**(1), 210–239 (2007)

Ambainis, A.: Quantum walks and their algorithmic applications. Int. J. Quantum Inf. **1**, 507–518 (2003)

Ambainis, A., Bach, E., Nayak, A., Vishwanath, A., Watrous, J.: One-dimensional quantum walks. In: Proc. STOC (2001)

Ambainis, A., Buhrman, H., Høyer, P., Karpinski, M., Kurur, P.: Quantum matrix verification. Unpublished manuscript (2002)

Ambainis, A., Kempe, J., Rivosh, A.: Coins make quantum walks faster. In: Proc. of SODA'05, pp 1099–1108

Ambainis, A., Kempe, J., Rivosh, A.: In: Proceedings of the ACM/SIAM Symposium on Discrete Algorithms (SODA'06), 2006, pp. 1099–1108

Ambainis, A., Mosca, M., Tapp, A., de Wolf, R.: Private quantum channels. In: Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science, 2000, pp. 547–553

Ambrosio, P., Auletta, V.: Deterministic monotone algorithms for scheduling on related machines. In: 2nd Ws. on Approx. and Online Alg. (WAOA), 2004, pp. 267–280

Ambühl, C.: Offline list update is NP-hard. In: Proc. 8th Annual European Symposium on Algorithms, pp. 42–51. LNCS, vol. 1879. Springer (2001)

Ambuhl, C., Erlebach, T., Mihalak, M., Nunkesser, M.: Constant-factor approximation for minimum-weight (connected) dominating sets in unit disk graphs. In: Proceedings of the 9th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX 2006), Barcelona, 28–30 August 2006, LNCS, vol. 4110, pp. 3–14. Springer, Berlin Heidelberg (2006)

Ambühl, C., Gärtner, B., von Stengel, B.: Towards new lower bounds for the list update problem. Theor. Comput. Sci **68**, 3–16 (2001)

Amenta, N., Bern, M.: Surface reconstruction by Voronoi filtering. Discret. Comput. Geom. **22**, 481–504 (1999)

Amenta, N., Bern, M., Eppstein, D.: The crust and the β-skeleton: Combinatorial curve reconstruction. Graph. Model. Image Process. **60**, 125–135 (1998)

Amir, A.: Theoretical issues of searching aerial photographs: a bird's eye view. Int. J. Found. Comput. Sci. **16**, 1075–1097 (2005)

Amir, A., Aumann, Y., Cole, R., Lewenstein, M., Porat, E.: Function matching: Algorithms, applications and a lower bound. In: Proc. of the 30th International Colloquium on Automata, Languages and Programming (ICALP), 2003 pp. 929–942

Amir, A., Benson, G.: Efficient two dimensional compressed matching. In: Proceeding of Data Compression Conference, Snow Bird, Utah, 1992, pp. 279–288

Amir, A., Benson, G.: Two-dimensional periodicity and its application. Proceeding of 3rd Symposium on Discrete Algorithms, Orlando, FL, 1992, pp. 440–452

Amir, A., Benson, G.: Two-dimensional periodicity and its application. SIAM J. Comput. **27**(1), 90–106 (1998)

Amir, A., Benson, G., Farach, M.: An alphabet independent approach to two dimensional pattern matching. SIAM J. Comput. **23**(2), 313–323 (1994)

Amir, A., Benson, G., Farach, M.: Let sleeping files lie: Pattern matching in Z-compressed files. J. Comput. Syst. Sci. **52**(2), 299–307 (1996)

Amir, A., Benson, G., Farach, M.: Optimal parallel two dimensional text searching on a crew pram. Inf. Comput. **144**(1), 1–17 (1998)

Amir, A., Benson, G., Farach, M.: Optimal two-dimensional compressed matching. J. Algorithms **24**(2), 354–379 (1997)

Amir, A., Benson, G., Farach, M.: The truth, the whole truth, and nothing but the truth: Alphabet independent two dimensional witness table construction.Technical Report GIT-CC-92/52, Georgia Institute of Technology (1992)

Amir, A., Butman, A., Crochemore, M., Landau, G.M., Schaps, M.: Two-dimensional pattern matching with rotations. Theor. Comput. Sci. **314**(1–2), 173–187 (2004)

Amir, A., Butman, A., Lewenstein, M.: Real scaled matching. Inf. Proc. Lett. **70**(4), 185–190 (1999)

Amir, A., Butman, A., Lewenstein, M., Porat, E.: Real two dimensional scaled matching. In: Proc. 8th Workshop on Algorithms and Data Structures (WADS '03), pp. 353–364 (2003)

Amir, A., Butman, A., Lewenstein, M., Porat, E., Tsur, D.: Efficient one dimensional real scaled matching. In: Proc. 11th Symposium on String Processing and Information Retrieval (SPIRE '04), pp. 1–9 (2004)

Amir, A., Calinescu, G.: Alphabet independent and dictionary scaled matching. J. Algorithms **36**, 34–62 (2000)

Amir, A., Chencinski, E.: Faster two dimensional scaled matching. In: Proc. 17th Annual Symposium on Combinatorial Pattern Matching. LNCS, vol. 4009, pp. 200–210. Springer, Berlin (2006)

Amir, A., Farach, M.: Two dimensional dictionary matching. Inf. Proc. Lett. **44**, 233–239 (1992)

Amir, A., Farach, M., Matias, Y.: Efficient randomized dictionary matching algorithms. In: Proc. of Symposium on Combinatorial Pattern Matching (CPM), 1992, pp. 259–272

Amir, A., Farach, M., Muthukrishnan, S.: Alphabet dependence in parameterized matching. Inf. Process. Lett. **49**, 111–115 (1994)

Amir, A., Kapah, O., Tsur, D.: Faster two dimensional pattern matching with rotations. In: Proc. 15th Annual Symposium on Combinatorial Pattern Matching. LNCS, vol. 3109, pp. 409–419. Springer, Berlin (2004)

Amir, A., Keselman, D.: Maximum agreement subtree in a set of evolutionary trees: Metrics and efficient algorithms. SIAM J. Comput. **26**(6), 1656–1669 (1997)

Amir, A., Keselman, D., Landau, G.M., Lewenstein, N., Lewenstein, M., Rodeh, M.: Indexing and dictionary matching with one error. In: Proc. of Workshop on Algorithms and Data Structures (WADS), 1999, pp. 181–192

Amir, A., Kopelowitz, T., Lewenstein, M., Lewenstein, N.: Towards real-time suffix tree construction. In: Proceedings of the 12th International Symposium on String Processing and Information Retrieval, SPIRE 2005. LNCS, vol. 3772, pp. 67–78. Springer, Berlin (2005)

Amir, A., Landau, G.: Fast parallel and serial multidimensional approximate array matching. Theor. Comput. Sci. **81**, 97–115 (1991)

Amir, A., Landau, G., Sokol, D.: Inplace 2d matching in compressed images. J. Algorithms **49**(2), 240–261 (2003)

Amir, A., Landau, G.M., Sokol, D.: Inplace run-length 2d compressed search. Theor. Comput. Sci. **290**(3), 1361–1383 (2003)

Amir, A., Landau, G.M., Vishkin, U.: Efficient pattern matching with scaling. J. Algorithms **13**(1), 2–32 (1992)

Amir, A., Lewenstein, M., Porat, E.: Faster algorithms for string matching with *k* mismatches. J. Algorithms **50**(2), 257–275 (2004)

Anagnostopoulos, A., Bent, R., Upfal, E., van Hentenryck, P.: A simple and deterministic competitive algorithm for online facility location. Inf. Comput. **194**(2), 175–202 (2004)

Anagnostou, E., Hadzilacos, V.: Tolerating Transient and Permanent Failures. In: Distributed Algorithms 7th International Workshop. LNCS, vol. 725, pp. 174–188. Springer, Heidelberg (1993)

Andelman, N., Azar, Y., Sorani, M.: Truthful approximation mechanisms for scheduling selfish related machines. In: 22nd Ann. Symp. on Theor. Aspects of Comp. Sci. (STACS), 2005, pp. 69–82

Andelman, N., Mansour, Y.: A sufficient condition for truthfulness with single parameter agents. In: Proc. 8th ACM Conference on Electronic Commerce (EC),Ann, Arbor, Michigan, June (2006)

Andelman, N., Mansour, Y., Zhu, A.: Competitive queueing policies in QoS switches. In: Proc. 14th Symp. on Discrete Algorithms (SODA), pp. 761–770 ACM/SIAM, San Francisco, CA, USA (2003)

Anderegg, L., Eidenbenz, S.: Ad hoc-VCG: a truthful and cost-efficient routing protocol for mobile ad hoc networks with selfish agents. In: Proceedings of the 9th annual international conference on Mobile computing and networking. pp. 245–259 ACM Press, New York (2003)

Anderson, E., Hall, J., Hartline, J., Hobbes, M., Karlin, A., Saia, J., Swaminathan, R., Wilkes, J.: An experimental study of data migration algorithms. In: Workshop on Algorithm Engineering (2001)

Anderson, E.J., Hildrum, K., Karlin, A.R., Rasala, A., Saks, M.: On list update and work function algorithms. Theor. Comput. Sci. **287**, 393–418 (2002)

Anderson, J.H.: Composite registers. Distrib. Comput. **6**, 141–154 (1993)

Anderson, J.H.: Multi-writer composite registers. Distrib. Comput. **7**, 175–195 (1994)

Anderson, J.H., Kim, Y.-J.: Adaptive mutual exclusion with local spinning. In: Proceedings of the 14th international symposium on distributed computing. Lect. Notes Comput. Sci. **1914**, 29–43, (2000)

Anderson, R.J.: The Role of Experiment in the Theory of Algorithms. In: Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 59, pp. 191–195. American Mathematical Society, Providence, RI (2002)

Anderson, T.E.: The performance of spin lock alternatives for shared-memory multiprocessor. IEEE Trans. Parallel Distrib. Syst. **1**(1), 6–16 (1990)

Andersson, A.: Faster deterministic sorting and searching in linear space. In: Proc. 37th FOCS, 1998, pp. 135–141

Andersson, A., Hagerup, T., Nilsson, S., Raman, R.: Sorting in linear time? J. Comp. Syst. Sci. **57**, 74–93 (1998). Announced at STOC'95

Andersson, A., Miltersen, P.B., Thorup, M.: Fusion trees can be implemented with AC^0 instructions only. Theor. Comput. Sci. **215**(1–2), 337–344 (1999)

Andersson, A., Nilsson, S.: A new efficient radix sort. In: Proceedings of the 35th Annual Symposium on Foundations of Computer Science (FOCS '94), IEEE Comput. Soc. Press, pp. 714–721 (1994)

Andersson, A., Nilsson, S.: Implementing radixsort. ACM J. Exp. Algorithmics **3**, 7 (1998)

Andersson, A., Thorup, M.: Dynamic ordered sets with exponential search trees. CoRR cs.DS/0210006. See also FOCS'96, STOC'00, 2002

Andersson, G., Engebretsen, L., Håstad, J.: A new way to use semidefinite programming with applications to linear equations mod *p*. J. Algorithms **39**, 162–204 (2001)

Andrews, M., Chuzhoy, J., Khanna, S., Zhang, L.: Hardness of the Undirected Edge-Disjoint Paths Problem with Congestion. Proc. of IEEE FOCS, 2005, pp. 226–244

Andrews, M., Zhang, L.: Hardness of the undirected congestion minimization problem. In: STOC '05: Proceedings of the thirty-seventh annual ACM symposium on Theory of computing, pp. 284–293. ACM Press, New York (2005)

Andrews, M., Zhang, L.: The access network design problem. In: Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp. 40–49. IEEE Computer Society, Los Alamitos, CA, USA (1998)

Aneja, Y.P.: An integer linear programming approach to the Steiner problem in graphs. Networks **10**(2), 167–178 (1980)

Angelopoulos, S., Dorrigiv, R., López-Ortiz, A.: On the separation and equivalence of paging strategies. In: Proceedings of the 18th Annual ACM–SIAM Symposium on Discrete Algorithms. ACM/SIAM, New York, Philadelphia (2007)

Angluin, D.: Learning regular sets from queries and counterexamples. Inf. Comput. **75**, 87–106 (1987)

Angluin, D.: Queries and concept learning. Mach. Learn. **2**(4), 319–342 (1988)

Angluin, D.: Queries Revisited. Theor. Comput. Sci. **313**(2), 175–194 (2004)

Angluin, D., Kharitonov, M.: When Won't Membership Queries Help? J. Comput. Syst. Sci. **50**, 336–355 (1995)

Angluin, D., Laird, P.: Learning from noisy examples. Mach. Learn. **2**, 343–370 (1988)

Anh, V.N., Moffat, A.: Improved word-aligned binary compression for text indexing. IEEE Trans. Knowl. Data Eng. **18**(6), 857–861 (2006)

Anthony, M., Bartlett, P.L.: Neural Network Learning: Theoretical Foundations. Cambridge University Press, Cambridge, England (1999)

Apostolico, A.: The myriad virtues of subword trees. In: Apostolico, A., Galil, Z. (eds.) Combinatorial Algorithms on Words. NATO ASI Series, vol. F12, pp. 85–96. Springer, Berlin (1985)

Apostolico, A., Erdös, P., Lewenstein, M.: Parameterized matching with mismatches. J. Discret. Algorithms **5**(1), 135–140 (2007)

Apostolico, A., Landau, G.M., Skiena, S.: Matching for Run Length Encoded Strings. J. Complex. **15**(1), 4–16 (1999)

Apostolico, A., Preparata, F.P.: Optimal off-line detection of repetitions in a string. Theor. Comput. Sci. **22**(3), 297–315 (1983)

Applegate, D., Bixby, R., Chvátal, V., Cook, W.: Finding tours in the TSP. Technical Report 99885, Research Institute for Discrete Mathematics, Universität Bonn (1999)

Applegate, D., Bixby, R., Chvátal, V., Cook, W.: On the solution of traveling salesman problems. Documenta Mathematica, Extra Volume Proceedings ICM III:645–656. Deutsche Mathematiker-Vereinigung, Berlin (1998)

Applegate, D., Cohen, E.: Making intra-domain routing robust to changing and uncertain traffic demands: understanding fundamental tradeoffs. In: SIGCOMM, pp. 313–324 (2003)

Applegate, D., Cohen, E.: Making routing robust to changing traffic demands: algorithms and evaluation. IEEE/ACM Trans Netw **14**(6), 1193–1206 (2006). doi:10.1109/TNET.2006.886296

Ar, S., Blum, M., Codenotti, B., Gemmell, P.: Checking approximate computations over the reals. In: Proceedings of the Twenty-Fifth Annual ACM Symposium on the Theory of Computing, pp. 786–795. ACM, New York (2003)

Arbell, O., Landau, G.M., Mitchell, J.: Edit Distance of Run-Length Encoded Strings. Inf. Proc. Lett. **83**(6), 307–314 (2002)

Archer, A.: Mechanisms for Discrete Optimization with Rational Agents. Ph. D. thesis, Cornell University (2004)

Archer, A., Papadimitriou, C.H., Talwar, K., Tardos, E.: An approximate truthful mechanism for combinatorial auctions with single parameter agents. In: Proc. 14th Ann. ACM–SIAM Symp. on Discrete Algorithms (SODA), pp. 205–214. Baltimore, Maryland (2003)

Archer, A., Tardos, É.: Truthful mechanisms for one-parameter agents. In: Proc. 42nd Annual Symposium on Foundations of Computer Science (FOCS), 2001, pp. 482–491

Arge, L.: The buffer tree: A technique for designing batched external data structures. Algorithmica **37**(1), 1–24 (2003)

Arge, L., Bender, M.A., Demaine, E.D., Holland-Minkley, B., Munro, J.I.: Cache-oblivious priority queue and graph algorithm applications. In: Proc. 34th Annual ACM Symposium on Theory of Computing, pp. 268–276. ACM Press, New York (2002)

Arge, L., Brodal, G.S., Fagerberg, R.: Cache-oblivious data structures. In: Mehta, D., Sahni, S. (eds.) Handbook on Data Structures and Applications. CRC Press, Boca Raton (2005)

Arge, L., Brodal, G.S., Fagerberg, R., Laustsen, M.: Cache-oblivious planar orthogonal range searching and counting. In: Proc. 21st ACM Symposium on Computational Geometry, pp. 160–169. ACM, New York (2005)

Arge, L., de Berg, M., Haverkort, H.J.: Cache-oblivious R-trees. In: Proc. 21st ACM Symposium on Computational Geometry, pp. 170–179. ACM, New York (2005)

Arge, L., de Berg, M., Haverkort, H.J., Yi, K.: The priority R-tree: A practically efficient and worst-case optimal R-tree. In: Proc. SIGMOD International Conference on Management of Data, 2004, pp. 347–358

Arge, L., Ferragina, P., Grossi, R., Vitter, J.S.: On sorting strings in external memory (extended abstract). In: Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC '97), ACM, ed., pp. 540–548. ACM Press, El Paso (1997),

Arge, L., Knudsen, M., Larsen, K.: A general lower bound on the I/O-complexity of comparison-based algorithms. In: Proceedings of the Workshop on Algorithms and Data Structures. Lect. Notes Comput. Sci. **709**, 83–94 (1993)

Arge, L., Samoladas, V., Vitter, J.S.: On two-dimensional indexability and optimal range search indexing. In: Proc. ACM Symposium on Principles of Database Systems, 1999, pp. 346–357

Arge, L., Samoladas, V., Yi, K.: Optimal external memory planar point enclosure. In: Proc. European Symposium on Algorithms, 2004

Arge, L., Vitter, J.S.: Optimal external memory interval management. SIAM J. Comput. **32**(6), 1488–1508 (2003)

Arge, L., Zeh, N.: Simple and semi-dynamic structures for cache-oblivious planar orthogonal range searching. In: Proc. 22nd ACM Symposium on Computational Geometry, pp. 158–166. ACM, New York (2006)

Arge, L.A.: External memory data structures. In: Abello, J., Pardalos, P.M., Resende, M.G.C. (eds.) Handbook of Massive Data Sets, pp. 313–357. Kluwer, Dordrecht (2002)

Arge, L.A., Hinrichs, K.H., Vahrenhold, J., Vitter, J.S.: Efficient bulk operations on dynamic R-trees. Algorithmica **33**, 104–128 (2002)

Arikati, S., Chen, D.Z., Chew, L.P., Das, G., Smid, M., Zaroliagis, C.D.: Planar spanners and approximate shortest path queries among obstacles in the plane. In: Proceedings of the 4th Annual European Symposium on Algorithms. Lecture Notes in Computer Science, vol. 1136, Berlin, pp. 514–528. Springer, London (1996)

Armon, A., Azar, Y., Epstein, L., Regev, O.: On-line restricted assignment of temporary tasks with unknown durations. Inf. Process. Lett. **85**(2), 67–72 (2003)

Armon, A., Azar, Y., Epstein, L., Regev, O.: Temporary tasks assignment resolved. Algorithmica **36**(3), 295–314 (2003)

Arnborg, S.: Efficient algorithms for combinatorial problems on graphs with bounded decomposability – A survey. BIT **25**, 2–23 (1985)

Arnborg, S., Corneil, D.G., Proskurowski, A.: Complexity of finding embeddings in a *k*-tree. SIAM J. Algebr. Discret. Methods **8**, 277–284 (1987)

Arnborg, S., Proskurowski, A.: Characterization and recognition of partial 3-trees. SIAM J. Algebr. Discret. Methods **7**, 305–314 (1986)

Arnold, R., Bell, T.: A corpus for the evaluation of lossless compression algorithms. In: Proceedings of the IEEE Data Compression Conference, Snowbird, Utah, March 1997, pp. 201–210

Aronov, B., de Berg, M., Cheong, O., Gudmundsson, J., Haverkort, H., Vigneron, A.: Sparse Geometric Graphs with Small Dilation. 16th International Symposium ISAAC 2005, Sanya. In: Deng, X., Du, D. (eds.) Algorithms and Computation, Proceedings. LNCS, vol. 3827, pp. 50–59. Springer, Berlin (2005)

Arora, S.: Approximation schemes for $\mathcal{NP}$-hard geometric optimization problems: A survey. Math. Program. Ser. B **97**, 43–69 (2003)

Arora, S.: Nearly linear time approximation schemes for Euclidean TSP and other geometric problems. In: Proc. 38th IEEE Symp. on Foundations of Computer Science, 1997, pp. 554–563

Arora, S.: Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. J. ACM **45**(5), 753–782 (1998)

Arora, S.: Polynomial-time approximation schemes for euclidean tsp and other geometric problem. J. ACM **45**, 753–782 (1998)

Arora, S.: Polynomial-time approximation schemes for Euclidean TSP and other geometric problems. In: Proc. 37th IEEE Symp. on Foundations of Computer Science, 1996, pp. 2–12

Arora, S., Chlamtac, E., Charikar, M.: New approximation guarantees for chromatic number. In: Proceedings of the 38th ACM Symposium on Theory of Computing (STOC), Seattle, May 2006, pp. 215–224

Arora, S., Grigni, M., Karger, D., Klein, P., Woloszyn, A..: A polynomial time approximation scheme for weighted planar graph TSP. In: Proc. 9th Annual ACM-SIAM Symposium on Discrete Algorithms, 1998, pp. 33–41

Arora, S., Hazan, E., Kale, S.: $O(\sqrt{\log n})$ approximation to sparsest cut in $\tilde{O}(n^2)$ time. In: FOCS '04: Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS'04), pp. 238–247. IEEE Computer Society, Washington (2004)

Arora, S., Kale, S.: A combinatorial, primal-dual approach to semidefinite programs. In: STOC '07: Proceedings of the 39th Annual ACM Symposium on Theory of Computing, pp. 227–236. ACM (2007)

Arora, S., Karger, D., Karpinski, M.: Polynomial time approximation schemes for dense instances of NP-hard problems. J. Comput. Syst. Sci. **58**(1), 193–210 (1999). Preliminary version in STOC 1995

Arora, S., Lee, J., Naor, A.: Euclidean Distortion and the Sparsest Cut. In: Proceedings of the 37th ACM Symposium on Theory of Computing (STOC), Baltimore, May 2005, pp. 553–562

Arora, S., Lee, J.R., Naor, A.: Euclidean distortion and the sparsest cut. In: STOC '05: Proceedings of the thirty-seventh annual ACM symposium on Theory of computing, pp. 553–562. ACM Press, New York (2005)

Arora, S., Lund, C., Motwani, R., Sudan, M., Szegedy, M.: Proof verification and the hardness of approximation problems. J. ACM **45**(3), 501–555 (1998)

Arora, S., Raghavan, P., Rao, S.: Approximation schemes for Euclidean *k*-medians and related problems. In: Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC), pp. 106–113. ACM, New York (1998)

Arora, S., Rao, S., Vazirani, U.: Expander Flows, Geometric Embeddings, and Graph Partitionings. In: Proceedings of the 36th ACM Symposium on Theory of Computing (STOC), Chicago, June 2004, pp. 222–231

Arora, S., Sudan, M.: Improved low degree testing and its applications. In: Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, pp. 485–495. ACM, New York (1997)

Arrow, K.J., Debreu, G.: Existence of an equilibrium for a competitive economy. Econometrica **22**(3), 265–290 (1954)

Arroyuelo, D., Navarro, G., Sadakane, K.: Reducing the space requirement of LZ-index. In: Proc. 17th Combinatorial Pattern Matching conference (CPM), LNCS no. 4009, pp. 318–329, Springer (2006)

Arslan A., Eğecioğlu, Ö, Pevzner, P.: A new approach to sequence comparison: normalized sequence alignment. Bioinformatics **17**, 327–337 (2001)

Arya, S., Das, G., Mount, D.M., Salowe, J.S., Smid, M.: Euclidean spanners: short, thin, and lanky. In: Proceedings of the 27th ACM Symposium on Theory of Computing, pp. 489–498. Las Vegas, 29 May–1 June 1995

Arya, S., Mount, D.M., Smid, M.: Dynamic algorithms for geometric spanners of small diameter: Randomized solutions. Comput. Geom. Theor. Appl. **13**(2), 91–107 (1999)

Arya, S., Mount, D.M., Smid, M.: Randomized and deterministic algorithms for geometric spanners of small diameter. In: Proceedings of the 35th IEEE Symposium on Foundations of Computer Science, pp. 703–712. Santa Fe, 20–22 November 1994

Arya, S., Smid, M.: Efficient construction of a bounded-degree spanner with low weight. Algorithmica **17**, 33–54 (1997)

Arya, V., Garg, N., Khandekar, R., Meyerson, A., Munagala, K., Pandit, V.: Local search heuristics for k-median and facility location problems. In: Proceedings of the 33rd Annual ACM Symposium on Theory of Computing (STOC), pp. 21–29. ACM, New York (2001)

Arya, V., Garg, N., Khandekar, R., Meyerson, A., Munagala, K., Pandit, V.: Local search heuristics for k-median and facility location problems. SIAM J. Comput. **33**(3), 544–562 (2004)

Asano, Y., Imai, H.: Practical efficiency of the linear-time algorithm for the single source shortest path problem. J. Oper. Res. Soc. Jpn. **43**(4), 431–447 (2000)

Aslam, J., Decatur, S.: Specification and simulation of statistical query algorithms for efficiency and noise tolerance. J. Comput. Syst. Sci. **56**, 191–208 (1998)

Aspnes, J.: Randomized protocols for asynchronous consensus. Distrib. Comput. **16**(2–3), 165–175 (2003)

Aspnes, J., Azar, Y., Fiat, A., Plotkin, S., Waarts, O.: On-line load balancing with applications to machine scheduling and virtual circuit routing. J. ACM **44**, 486–504 (1997)

Aspnes, J., Azar, Y., Fiat, A., Plotkin, S., Waarts, O.: On-line routing of virtual circuits with applications to load balancing and machine scheduling. J. ACM **44**(3), 486–504 (1997)

Aspnes, J., Herlihy, M.: Wait-free data structures in the asynchronous PRAM model. In: Proc. 2nd ACM Symposium on Parallel Algorithms and Architectures, Crete, July 1990. pp. 340–349. ACM, New York, 1990

Aspnes, J., Shah, G.: Skip graphs. In: Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, Baltimore, January 2003, pp. 384–393

Aspnes, J., Waarts, O.: Randomized consensus in expected $o(n \log^2 n)$ operations per processor. In: Proceedings of the 33rd Symposium on Foundations of Computer Science. 24–26 October 1992, pp. 137–146. IEEE Computer Society, Pittsburgh (1992)

Aspvall, B., Plass, M.F., Tarjan R.E.: A linear-time algorithm for testing the truth of certain quantified boolean formulas. Inf. Proc. Lett. **8**(3), 121–123 (1979)

Atici, A., Servedio, R.A.: Learning unions of $\omega(1)$-dimensional rectangles. In: Proceedings of 17th Algorithmic Learning Theory Conference, pp. 32–47. Springer, New York (2006)

Atici, A., Servedio, R.A.: Learning unions of $\omega(1)$-dimensional rectangles. In: ALT, pp. 32–47 (2006)

Atkins, J.E., Middendorf, M.: On physical mapping and the consecutive ones property for sparse matrices. Discret. Appl. Math. **71**(1–3), 23–40 (1996)

Atkinson, M.D.: An optimal algorithm for geometric congruence. J. Algorithms **8**, 159–172 (1997)

Attallah, M., Callahan, P., Goodrich, M.: P-complete geometric problems. Int. J. Comput. Geom. Appl. **3**(4), 443–462 (1993)

Atteson, K.: The performance of neighbor-joining methods of phylogenetic reconstruction. Algorithmica **25**, 251–278 (1999)

Attiya, H.: Efficient and robust sharing of memory in message-passing systems. J. Algorithms **34**(1), 109–127 (2000)

Attiya, H., Bar-Noy, A., Dolev, D.: Sharing memory robustly in message-passing systems. J. ACM **42**(1), 124–142 (1995)

Attiya, H., Bar-Noy, A., Dolev, D., Peleg, D., Reischuk, R.: Renaming in an asynchronous environment. J. ACM **37**(3), 524–548 (1990)

Attiya, H., Censor, K.: Tight bounds for asynchronous randomized consensus. In: Proceedings of the Symposium on the Theory of Computation. San Diego, 11–13 June 2007 ACM Special Interest Group on Algorithms and Computation Theory (SIGACT) (2007)

Attiya, H., Fouren, A.: Adaptive and efficient algorithms for lattice agreement and renaming. SIAM J. Comput. **31**, 642–664 (2001)

Attiya, H., Fouren, A., Gafni, E.: An adaptive collect algorithm with applications. Distrib. Comput. **15**, 87–96 (2002)

Attiya, H., Guerraoui, R., Hendler, D., Kouznetsov, P.: Synchronizing without locks is inherently expensive. In: PODC '06: Proceedings of the twenty-fifth Annual ACM Symposium on Principles of Distributed Computing, New York, USA, pp. 300–307. ACM Press (2006)

Attiya, H., Guerraoui, R., Kouznetsov, P.: Computing with reads and writes in the absence of step contention. In: Proc. 19th Annual International Symposium on Distributed Computing, 2005

Attiya, H., Herlihy, M., Rachman, O.: Atomic snapshots using lattice agreement. Distrib. Comput. **8**, 121–132 (1995)

Attiya, H., Herzberg, A., Rajsbaum, S.: Optimal clock synchronization under different delay assumptions. SIAM J. Comput. **25**(2), 369–389 (1996)

Attiya, H., Rachman, O.: Atomic snapshots in $O(n \log n)$ operations. SIAM J. Comput. **27**, 319–340 (1998)

Attiya, H., Welch, J.: Distributed Computing: Fundamentals, Simulations and Advanced Topics, 2nd edn. Wiley-Interscience, Hoboken (2004)

Attiya, H., Welch, J.L.: Distributed Computing: Fundamentals, Simulations and Advanced Topics. McGraw-Hill, UK (1998)

Audsley, N., Burns, A., Wellings, A.: Deadline monotonic scheduling theory and application. Control Eng. Pract. **1**, 71–78 (1993)

Auer, P., Cesa-Bianchi, N.: On-line learning with malicious noise and the closure algorithm. Ann. Math. Artif. Intell. **23**, 83–99 (1998)

Auer, P., Warmuth, M.K.: Tracking the best disjunction. Mach. Learn. **32**(2), 127–150 (1998)

Auletta, V., De Prisco, R., Penna, P., Persiano, G.: Deterministic truthful approximation mechanisms for scheduling related machines. In: 21st Ann. Symp. on Theor. Aspects of Comp. Sci. (STACS), 2004, pp. 608–619

Auletta, V., De Prisco, R., Penna, P., Persiano, G., Ventre, C.: New constructions of mechanisms with verification. In: 33rd International Colloquium on Automata, Languages and Programming (ICALP) (1), 2006, pp. 596–607

Aumann, Y.: Efficient asynchronous consensus with the weak adversary scheduler. In: Symposium on Principles of Distrib. Comput.(PODC) Santa Barbara, 21–24 August 1997, pp. 209–218. ACM Special Interest Group on Algorithms and Computation Theory (SIGACT) (1997)

Aumann, Y., Kapach-Levy, A.: Cooperative sharing and asynchronous consensus using single-reader/single-writer registers. In: Proceedings of 10th Annual ACM-SIAM Symposium of Discrete Algorithms (SODA) Baltimore, 17–19 January 1999, pp. 61–70. Society for Industrial and Applied Mathematics (SIAM) (1999)

Aumann, Y., Rabani, Y.: An O(log k) approximate min-cut max-flow theorem and approximation algorithm. SIAM J. Comput. **27**(1), 291–301 (1998)

Ausiello, G., Crescenzi, P., Gambosi, G., Kann, V., Marchetti-Spaccamela, A., Protasi, M.: Complexity and approximation: combinatorial optimization problems and their approximability properties. Springer, Berlin (1999)

Ausiello, G., Italiano, G.F., Marchetti-Spaccamela, A., Nanni, U.: Incremental algorithms for minimal length paths. J. Algorithm **12**(4), 615–38 (1991)

Auslander, L., Parter, S.V.: On imbedding graphs in the plane. J. Math. and Mech. **10**, pp. 517–523 (1961)

Ausubel, L.M., Milgrom, P.R.: Ascending auctions with package bidding. Front. Theor. Econ. **1**(1) Article 1 (2002)

Avrahami, N., Azar, Y.: Minimizing total flow time and completion time with immediate dispacthing. In: Proceedings of 15th SPAA, pp. 11–18. (2003)

Avram, F., Bertsimas, D., Ricard, M.: Fluid models of sequencing problems in open queueing networks: an optimal control approach. In: Kelly, F.P., Williams, R.J. (eds.) Stochastic Networks. Proceedings of the International Mathematics Association, vol. 71, pp. 199–234. Springer, New York (1995)

Awerbuch, B.: Complexity of network synchronization. J. ACM **4**, 804–823 (1985)

Awerbuch, B.: Optimal distributed algorithms for minimum weight spanning tree, counting, leader election and related problems (detailed summary). In: Proc. of the 19th Annual ACM Symposium on Theory of Computing, pp. 230–240. ACM, USA (1987)

Awerbuch, B., Azar, Y., Bartal, Y.: On-line generalized Steiner problem. In: Proc. of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms, Society for Industrial and Applied Mathematics, Philadelphia, 2005, pp. 68–74 (1996)

Awerbuch, B., Azar, Y., Epstein A.: Large the price of routing unsplittable flow. In: Proc. of the 37th Annual ACM Symposium on Theory of Computing (STOC), pp. 57–66. ACM, Baltimore (2005)

Awerbuch, B., Azar, Y., Leonardi, S., Regev, O.: Minimizing the flow time without migration. SIAM J. Comput. **31**, 1370–1382 (2002)

Awerbuch, B., Azar, Y., Meyerson, A.: Reducing truth-telling online mechanisms to online optimization. In: Proc. 35th Ann. ACM. Symp. on Theory of Comput. (STOC), San Diego, California (2003)

Awerbuch, B., Azar, Y., Richter, Y., Tsur, D.: Tradeoffs in worst-case equilibria. In: Approximation and Online Algorithms, 1st International Workshop (WAOA), pp. 41–52. Springer, Budapest (2003)

Awerbuch, B., Azar, Y., Richter, Y., Tsur, D.: Tradeoffs in worst-case equlibria. Theor. Comput. Sci. **361**, 200–209 (2006)

Awerbuch, B., Baratz, A., Peleg, D.: Efficient broadcast and light weight spanners. Tech. Report CS92-22, Weizmann Institute of Science (1992)

Awerbuch, B., Berger, B., Cowen, L., Peleg D.: Near-linear time construction of sparse neighborhood covers. SIAM J. Comput. **28**, 263–277 (1998)

Awerbuch, B., Patt-Shamir, B., Peleg, D., Saks, M.E.: Adapting to asynchronous dynamic networks. In: Proc. of the 24th Annual

ACM Symp. on Theory of Computing, Victoria, 4–6 May 1992, pp. 557–570

Awerbuch, B., Peleg, D.: Network synchronization with polylogarithmic overhead. In: Proc. 31st IEEE Symp. on Foundations of Computer Science, Sankt Louis, 22–24 Oct. 1990, pp. 514–522

Awerbuch, B., Peleg, D.: Routing with polynomial communication-space tradeoff. SIAM J. Discret. Math. **5**, 151–162 (1992)

Aydin, H., Melhem, R., Mosse, D., Alvarez, P.M.: Determining Optimal Processor Speeds for Periodic Real-Time Tasks with Different Power Characteristics. Euromicro Conference on Real-Time Systems, pp. 225–232. IEEE Computer Society, Washington, DC, USA (2001)

Azar, Y., Broder, A.Z., Karlin, A.R.: On-line load balancing. Theor. Comput. Sci. **130**, 73–84 (1994)

Azar, Y., Broder, A.Z., Karlin, A.R., Upfal, E.: Balanced allocations. SIAM J. Comput. **29**(1), 180–200 (1999)

Azar, Y., Chaiutin, Y.: Optimal node routing. In: Proceedings of the 23rd International Symposium on Theoretical Aspects of Computer Science, 2006, pp. 596–607

Azar, Y., Cohen, E., Fiat, A., Kaplan, H., Räcke, H.: Optimal oblivious routing in polynomial time. In: Proceedings of the 35th ACM Symposium on the Theory of Computing, pp. 383–388 (2003)

Azar, Y., Epstein, L.: On-line load balancing of temporary tasks on identical machines. SIAM J. Discret. Math. **18**(2), 347–352 (2004)

Azar, Y., Epstein, L., van Stee, R.: Resource augmentation in load balancing. J. Sched. **3**(5), 249–258 (2000)

Azar, Y., Fiat, A., Levy, M., Narayanaswamy, N.S.: An improved algorithm for online coloring of intervals with bandwidth. Theor. Comput. Sci. **363**(1), 18–27 (2006)

Azar, Y., Gamzu, I., Gutner, S.: Truthful unsplittable flow for large capacity networks. In: Proc. 19th Ann. ACM Symp. on Parallelism in Algorithms and Architectures (SPAA), pp. 320–329 (2007)

Azar, Y., Kalyanasundaram, B., Plotkin, S., Pruhs, K., Waarts, O.: Online load balancing of temporary tasks. J. Algorithms **22**(1), 93–110 (1997)

Azar, Y., Litichevskey, M.: Maximizing throughput in multi-queue switches. In: Proc. 12th Annual European Symp. on Algorithms (ESA), 53–64 (2004)

Azar, Y., Naor, J., Rom, R.: The competitiveness of on-line assignments. J. Algorithms **18**, 221–237 (1995)

Azar, Y., Regev, O.: Combinatorial algorithms for the unsplittable flow problem. Algorithmica **44**(1), 49–66 (2006). Preliminary version in Proc. of IPCO 2001

Azar, Y., Richter, Y.: An improved algorithm for CIOQ switches. In: Proc. 12th Annual European Symp. on Algorithms (ESA). LNCS, vol. 3221, 65–76 (2004)

Azar, Y., Richter, Y.: Management of multi-queue switches in QoS Networks. In: Proc. 35th ACM Symp. on Theory of Computing (STOC), 82–89 (2003)

Azar, Y., Richter, Y.: The zero-one principle for switching networks. In: Proc. 36th ACM Symp. on Theory of Computing (STOC), 64–71 (2004)

Aziz, A., Tasiran, S., Brayton, R.: BDD Variable Ordering for Interacting Finite State Machines. In: ACM Design Automation Conference, pp. 283–288. (1994)

Babai, L.: On Lovasz' lattice reduction and the nearest lattice point problem. Combinatorica **6**(1), 1–13 (1986). Preliminary version in STACS 1985

Babai, L., Luks, E.: Canonical labelling of graphs. In: Proceedings of the 15th Annual ACM Symposium on Theory of Computing, pp. 171–183. ACM, New York (1983)

Babaioff, M., Lavi, R., Pavlov, E.: Single-value combinatorial auctions and implementation in undominated strategies. In: Proc. of the 17th Symposium on Discrete Algorithms (SODA), 2006

Babcock, B., Babu, S., Datar, M., Motwani, R., Widom, J.: Models and issues in data stream systems. In: Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, pp. 1–16. ACM Press (2002)

Bachrach, B., El-Yaniv, R., Reinstädtler, M.: On the competitive theory and practice of online list accessing algorithms. Algorithmica **32**, 201–245 (2002)

Bader, D.A., Moret, B.M.E., Sanders, P.: Algorithm engineering for parallel computation. In: Fleischer, R., Meineche-Schmidt, E., Moret, B.M.E. (ed) Experimental Algorithmics. Lecture Notes in Computer Science, vol. 2547, pp. 1–23. Springer, Berlin (2002)

Bader, D.A., Moret, B.M.E., Vawter, L.: Industrial applications of high-performance computing for phylogeny reconstruction. In: Siegel, H.J. (ed.) Proc. SPIE Commercial Applications for High-Performance Computing, vol. 4528, pp. 159–168, Denver, CO (2001)

Bader, D.A., Moret, B.M.E., Warnow, T., Wyman, S.K., Yan, M.: High-performance algorithm engineering for gene-order phylogenies. In: DIMACS Workshop on Whole Genome Comparison, Rutgers University, Piscataway, NJ (2001)

Bader, D.A., Moret, B.M.E., Yan, M.: A linear-time algorithm for computing inversion distance between signed permutations with an experimental study. J. Comput. Biol. **8**(5), 483–491 (2001) An earlier version of this work appeared In: the Proc. 7th Int'l Workshop on Algorithms and Data Structures (WADS 2001)

Badimo, A., Bergheim, A., Hazelhurst, S., Papathanasopolous, M., Morris, L.: The stability of phylogenetic tree construction of the HIV-1 virus using genome-ordering data versus env gene data. In: Proc. ACM Ann. Research Conf. of the South African institute of computer scientists and information technologists on enablement through technology (SAICSIT 2003), vol. 47, pp. 231–240, Fourways, ACM, South Africa, September 2003

Bae, S.E., Takaoka, T.: Algorithms for the problem of k maximum sums and a VLSI algorithm for the k maximum subarrays problem. Proceedings of the 7th International Symposium on Parallel Architectures, Algorithms and Networks, pp. 247–253 (2004)

Baeza-Yates, R., Navarro, G.: Faster approximate string matching. Algorithmica **23**(2), 127–158 (1999)

Baeza-Yates, R., Navarro, G.: New models and algorithms for multidimensional approximate pattern matching. J. Discret. Algorithms **1**, 21–49 (2000)

Baeza-Yates, R., Schott, R.: Parallel searching in the plane. Comput. Geom. Theor. Appl. **5**, 143–154 (1995)

Baeza-Yates, R.A., Culberson, J.C., Rawlins, G.J.E.: Searching in the plane. Inf. Comput. **106**(2), 234–252 (1993)

Bafna, V., Berman, P., Fujito, T.: A 2-approximation algorithm for the undirected feedback vertex set problem. SIAM J. Discret. Math. **3**(2), 289–297 (1999)

Bafna, V., Gusfield, D., Lancia, G., Yooseph, S.: Haplotyping as perfect phylogeny: a direct approach. J. Comput. Biol. **10**(3–4), 323–340 (2003)

Bafna, V., Lawler, E.L., Pevzner, P.A.: Approximation algorithms for multiple sequence alignment. Theor. Comput. Sci. **182**, 233–244 (1997)

Bafna, V., Pevzner, P.A.: Genome rearrangements and sorting by reversals. SIAM J. Comput. **25**, 272–289 (1996)

Bafna, V., Pevzner, P.A.: Sorting by Transpositions. SIAM J. Discret. Math. **11**(2), 224–240 (1998)

Baïou, M., Balinski, M.: Erratum: The Stable Allocation (or Ordinal Transportation) Problem. Math. Oper. Res. **27**, 662–680 (2002)

Baïou, M., Balinski, M.: Student admissions and faculty recruitment. Theor. Comput. Sci. **322**(2), 245–265 (2004)

Baker, B.S.: A theory of parameterized pattern matching: algorithms and applications. In: Proc. 25th Annual ACM Symposium on the Theory of Computation (STOC), 1993, pp. 71–80

Baker, B.S.: Approximation algorithms for NP-complete problems on planar graphs. J. Assoc. Comput. Mach. **41**(1), 153–180 (1994)

Baker, B.S.: Parameterized diff. In: Proc. 10th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 1999, pp. 854–855

Baker, B.S.: Parameterized duplication in strings: Algorithms and an application to software maintenance. SIAM J. Comput. **26**(5), 1343–1362 (1997)

Baker, B.S.: Parameterized pattern matching by Boyer-Moore-type algorithms. In: Proc. 6th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 1995, pp. 541–550

Baker, B.S.: Parameterized pattern matching: Algorithms and applications. J. Comput. Syst. Sci. **52**(1), 28–42 (1996)

Baker, J., Cruz, I., Liotta, G., Tamassia, R.: A New Model for Algorithm Animation over the WWW, ACM Comput. Surv. **27**, 568–572 (1996)

Baker, J., Cruz, I., Liotta, G., Tamassia, R.: Animating Geometric Algorithms over the Web. In: Proceedings of the 12th Annual ACM Symposium on Computational Geometry. Philadelphia, Pennsylvania, May 24–26, pp. C3–C4 (1996)

Baker, J., Cruz, I., Liotta, G., Tamassia, R.: The Mocha Algorithm Animation System. In: Proceedings of the 1996 ACM Workshop on Advanced Visual Interfaces. Gubbio, Italy, May 27–29, pp. 248–250 (1996)

Baker, R., Boilen, M., Goodrich, M., Tamassia, R., Stibel, B.: Testers and Visualizers for Teaching Data Structures. In: Proceeding of the 13th SIGCSE Technical Symposium on Computer Science Education. New Orleans, March 24–28, pp. 261–265 (1999)

Baker, T.P.: A technique for extending rapid exact-match string matching to arrays of more than one dimension. SIAM J. Comput. **7**, 533–541 (1978)

Baker, T.P.: Stack-based scheduling of real-time processes. Real-Time Systems: The Int. J. Time-Critical Comput. **3**, 67–100 (1991)

Balcan, M., Blum, A., Hartline, J., Mansour, Y.: Mechanism design via machine learning. In: Proc. of the 46th Annual Symposium on Foundations of Computer Science (FOCS'05), 2005

Balcázar, J.L., Castro, J., Guijarro, D.: A new abstract combinatorial dimension for exact learning via queries. J. Comput. Syst. Sci. **64**(1), 2–21 (2002)

Balcázar, J.L., Castro, J., Guijarro, D., Simon, H.-U.: The consistency dimension and distribution-dependent learning from queries. Theor. Comput. Sci. **288**(2), 197–215 (2002)

Balinski, M.L.: On finding integer solutions to linear programs. In: Proceedings of the IBM Scientific Computing Symposium on Combinatorial Problems, pp. 225–248 IBM, White Plains, NY (1966)

Balinski, M.L., Wolfe, P.: On Benders decomposition and a plant location problem. In ARO-27. Mathematica Inc. Princeton (1963)

Ballester, C.: NP-completeness in Hedonic Games. Games. Econ. Behav. **49**(1), 1–30 (2004)

Banerjee, S., Konishi, H., Sönmez, T.: Core in a simple coalition formation game. Soc. Choice. Welf. **18**, 135–153 (2001)

Bansal, N.: Minimizing flow time on a constant number of machines with preemption. Oper. Res. Lett. **33**, 267–273 (2005)

Bansal, N., Blum, A., Chawla, S.: Meyerson, A.: Online oblivious routing. In: Proceedings of the 15th Annual ACM Symposium on Parallel Algorithms, 2003, pp. 44–49

Bansal, N., Blum, A., Chawla, S., Meyerson, A.: Online oblivious routing. In Symposium on Parallelism in Algorithms and Architectures, pp. 44–49 (2003)

Bansal, N., Buchbinder, N., Naor, J.: A primal-dual randomized algorithm for weighted paging. Proceedings of 48th Annual IEEE Symposium on Foundations of Computer Science, pp. 507–517 (2007)

Bansal, N., Dhamdhere, K., Könemann, J., Sinha, A.: Non-Clairvoyant Scheduling for Minimizing Mean Slowdown. Algorithmica **40**(4), 305–318 (2004)

Bansal, N., Fleischer, L., Kimbrel, T., Mahdian, M., Schieber, B., Sviridenko, M.: Further improvements in competitive guarantees for QoS buffering. In: Proc. 31st International Colloquium on Automata, Languages, and Programming (ICALP). Lecture Notes in Computer Science, vol. 3142, pp. 196–207. Springer, Berlin (2004)

Bansal, N., Kimbrel, T., Pruhs, K.: Speed scaling to manage energy and temperature. J. ACM **54**(1) (2007)

Bansal, N., Pruhs, K.: Server scheduling in the Lp norm: a rising tide lifts all boat. In: Symposium on Theory of Computing, STOC, pp. 242–250 (2003)

Bansal, N., Pruhs, K.: Server scheduling in the weighted Lp norm. In: LATIN, pp. 434–443 (2004)

Bansal, N., Pruhs, K., Stein, C.: Speed scaling for weighted flow. In: ACM/SIAM Symposium on Discrete Algorithms, 2007

Bansal, N., Raman, V.: Upper bounds for Max Sat: Further Improved. In: Proceedings of ISAAC. LNCS, vol. 1741, pp. 247–258. Springer, Berlin (1999)

Bansal, V., Agrawal, A., Malhotra, V.S.: Stable marriages with multiple partners: efficient search for an optimal solution. In: Proceedings of ICALP '03: the 30th International Colloquium on Automata, Languages and Programming. Lecture Notes in Computer Science, vol. 2719, pp. 527–542. Springer, Berlin (2003)

Bansalm, N., Kimbrel, T., Pruhs, K.: Dynamic Speed Scaling to Manage Energy and Temperature, Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science, pp. 520–529. IEEE Computer Society, Washington, DC, USA (2004)

Bao, L., Garcia—Aceves, J.J.: Topology management in ad hoc networks. In: Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing, Annapolis, 1–3 June 2003, pp. 129–140. ACM Press, New York (2003)

Bar-Eli, E., Berman, P., Fiat, A., Yan, P.: Online navigation in a room. J. Algorithms **17**(3), 319–341 (1994)

Bar-Noy, A., Dolev, D., Dwork, C., Strong, H.R.: Shifting Gears: Changing Algorithms on the Fly To Expedite Byzantine Agreement. In: PODC, 1987, pp. 42–51

Bar-Noy, A., Freund, A., Naor, J.: New algorithms for related machines with temporary jobs. J. Sched. **3**(5), 259–272 (2000)

Bar-Noy, A., Freund, A., Naor, J.: On-line load balancing in a hierarchical server topology. SIAM J. Comput. **31**, 527–549 (2001)

Bar-Noy, A., Motwani, R., Naor, J.: The greedy algorithm is optimal for on-line edge coloring. Inf. Proc. Lett. **44**(5), 251–253 (1992)

Bar-Yehuda, R., Even, S.: A local-ratio theorem for approximating the weighted vertex cover problem. Ann. Discret. Math. **25**, 27–45 (1985)

Bar-Yehuda, R., Geiger, D., Naor, J., Roth, R.M.: Approximation algorithms for the feedback vertex set problem with applications to constraint satisfaction and Bayesian inference. SIAM J. Comput. **27**(4), 942–959 (1998)

Bar-Yehuda, R., Goldreich, O., Itai, A.: On the time-complexity of broadcast in multi-hop radio networks: An exponential gap between determinism and randomization. J. Comput. Syst. Sci. **45**(1), 104–126 (1992)

Bar-Yehuda, R., Halldorsson, M., Naor, J., Shachnai, H., Shapira, I.: Scheduling split intervals. In: Proc. 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 2002, pp. 732–741

Bar-Yossef, Z., Hildrum, K., Wu, F.: Incentive-compatible online auctions for digital goods. In: Proceedings of the 13th Annual ACM-SIAM Symposium On Discrete Mathematics (SODA-02), New York, 6–8 January 2002, pp. 964–970. ACM Press, New York (2002)

Barahona, F.: On cuts and matchings in planar graphs. Math. Program. **60**, 53–68 (1993)

Barbay, J., Golynski, A., Munro, J.I., Rao, S.S.: Adaptive searching in succinctly encoded binary relations and tree-structured documents. In: Proceedings of the 17th Annual Symposium on Combinatorial Pattern Matching (CPM). Lecture Notes in Computer Science (LNCS), vol. 4009, pp. 24–35. Springer, Berlin (2006)

Barbay, J., Golynski, A., Munro, J.I., Rao, S.S.: Adaptive searching in succinctly encoded binary relations and tree-structured documents. In: Proc. 17th Combinatorial Pattern Matching (CPM). LNCS n. 4009 Springer, Barcelona (2006), pp. 24–35

Barbay, J., He, M., Munro, J.I., Rao, S.S.: Succinct indexes for strings, binary relations and multi-labeled trees. In: Proc. 18th ACM-SIAM Symposium on Discrete Algorithms (SODA), New Orleans, USA, (2007), pp. 680–689

Barber, C.B., Dobkin, D.P., Huhdanpaa, H.T.: Imprecision in QHULL. http://www.qhull.org/html/qh-impre.htm. Accessed 6 Apr 2008

Barborak, M., Dahbura, A., Malek, M.: The Consensus Problem in Fault-Tolerant Computing. ACM Comput. Surv. **25**(2), 171–220 (1993)

Barequet, G., Har-Peled, S.: Polygon containment and translational min-hausdorff-distance between segment sets are 3SUM-hard. Int. J. Comput. Geom. Appl. **11**(4), 465–474 (2001)

Barr, A., Feigenbaum, E.A.: The Handbook of Artificial Intelligence. Addison-Wesley Pub (Sd) (1994)

Barr, R.S., Golden, B.L., Kelly, J.P., Resende, M.G.C., Stewart, W.R.: Designing and reporting on computational experiments with heuristic methods. J. Heuristic **1**(1), 9–32 (1995)

Barrett, C., Bissett, K., Holzer, M., Konjevod, G., Marathe, M., Wagner, D.: Implementations of routing algorithms for transportation networks. In: 9th DIMACS Implementation Challenge Workshop: Shortest Paths. DIMACS Center, Piscataway, NJ, 13–14 Nov 2006

Barrett, C.L., Bisset, K., Jacob, R., Konjevod, G., Marathe, M.V.: Classical and contemporary shortest path problems in road networks: Implementation and experimental analysis of the TRANSIMS router. In: Algorithms – ESA 2002: 10th Annual European Symposium, Rome, Italy, 17–21 September 2002. Lecture Notes Computer Science, vol. 2461, pp. 126–138. Springer, Berlin (2002)

Barrière, L., Fraigniaud, P., Narayanan, L.: Robust Position-Based Routing in Wireless Ad Hoc Networks with Unstable Transmission Ranges. In: Proc. of the 5th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DIAL-M), pp 19–27. ACM Press, New York (2001)

Bartal, Y.: On approximating arbitrary metrices by tree metrics. In: STOC '98: Proceedings of the thirtieth annual ACM symposium on Theory of computing, pp. 161–168. ACM Press, New York (1998)

Bartal, Y.: Probabilistic approximation of metric spaces and its algorithmic applications. In: FOCS '96: Proceedings of the 37th Annual Symposium on Foundations of Computer Science, Washington, DC, USA, IEEE Computer Society, pp. 184–193 (1996)

Bartal, Y., Blum, A., Burch, C., Tomkins, A.: A polylog()-competitive algorithm for metrical task systems. In: Proceedings of the 29th annual ACM Symposium on the Theory of Computing, pp. 711–719. ACM, New York (1997)

Bartal, Y., Bollobás, B., Mendel, M.: Ramsey-type theorems for metric spaces with applications to online problems. J. Comput. Syst. Sci. **72**, 890–921 (2006)

Bartal, Y., Byers, J.W., Raz, D.: Global optimization using local information with applications to flow control. In: Proc. of the 38th IEEE Symposium on the Foundations of Computer Science (FOCS), pp. 303–312 (1997)

Bartal, Y., Charikar, M., Raz, D.: Approximating min-sum k-clustering in metric spaces. In: STOC '01: Proceedings of the thirty-third annual ACM symposium on Theory of computing, pp. 11–20. ACM Press, New York (2001)

Bartal, Y., Fiat, A., Karloff, H., Vohra, R.: New algorithms for an ancient scheduling problem. J. Comput. Syst. Sci. **51**(3), 359–366 (1995)

Bartal, Y., Gonen, R., Nisan, N.: Incentive compatible multi-unit combinatorial auctions. In: Proc. of the 9th Conference on Theoretical Aspects of Rationality and Knowledge (TARK'03), 2003

Bartal, Y., Mendel, M.: Multiembedding of metric spaces. SIAM J. Comput. **34**, 248–259 (2004)

Barthel, W., Hartmann, A.K., Leone, M., Ricci-Tersenghi, F., Weigt, M., Zecchina, R.: Hiding solutions in random satisfiability problems: A statistical mechanics approach. Phys. Rev. Lett. **88**, 188701 (2002)

Barve, R.D., Kallahalla, M., Varman, P.J., Vitter, J.S.: Competitive analysis of buffer management algorithms. J. Algorithms **36**, 152–181 (2000)

Barve, R.D., Vitter, J.S.: A simple and efficient parallel disk mergesort. ACM Trans. Comput. Syst. **35**, 189–215 (2002)

Basch, J.: Kinetic Data Structures. Ph. D. thesis, Stanford University (1999)

Basch, J., Guibas, L., Hershberger, J.: Data structures for mobile data. J. Algorithms **31**, 1–28 (1999)

Bast, H., Funke, S., Matijevic, D.: Transit: Ultrafast shortest-path queries with linear-time preprocessing. In: 9th DIMACS Implementation Challenge Workshop: Shortest Paths, DIMACS Center, Piscataway, NJ, 13–14 Nov 2006

Bast, H., Funke, S., Matijevic, D., Sanders, P., Schultes, D.: In transit to constant time shortest-path queries in road networks. In: Workshop on Algorithm Engineering and Experiments, 2007, pp. 46–59

Bast, H., Funke, S., Sanders, P., Schultes, D.: Fast routing in road networks with transit nodes. Science **316**(5824), 566 (2007)

Baswana, S., Sen, S.: A simple and linear time randomized algorithm for computing sparse spanners in weighted graphs. Random Struct. Algorithms **30**, 532–563 (2007)

Baswana, S., Sen, S.: Approximate distance oracles for unweighted graphs in $\tilde{O}(n^2)$ time. In: Proceedings of the 15th ACM-SIAM Symposium on Discrete Algorithms, pp. 271–280. ACM Press, New York (2004)

Baswana, S., Telikepalli, K., Mehlhorn, K., Pettie, S.: New construction of (α, β)-spanners and purely additive spanners. In: Proceedings of 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 2005, pp. 672–681

Batagelj, V., Pisanski, T., Simões-Pereira, J.M.S.: An algorithm for tree-realizability of distance matrices. Int. J. Comput. Math. **34**, 171–176 (1990)

Batu, T., Ergün, F., Sahinalp, S.C.: Oblivious string embeddings and edit distance approximations. Proc. ACM-SIAM SODA 792–801 (2006)

Baugh, J., Moussa, O., Ryan, C.A., Nayak, A., Laflamme, R.: Experimental implementation of heat-bath algorithmic cooling using solid-state nuclear magnetic resonance. Nature **438**, 470–473 (2005)

Baumer, S., Schuler, R.: Improving a probabilistic 3-SAT algorithm by dynamic search and independent clause pairs. ECCC TR03-010, (2003) Also presented at SAT (2003)

Baumer, S., Schuler, R.: Improving a Probabilistic 3-SAT Algorithm by Dynamic Search and Independent Clause Pairs. In: SAT 2003, pp. 150–161

Baur, C., Fekete, S.P.: Approximation of geometric dispersion problems. Algorithmica **30**(3), 451–470 (2001)

Bayer, R., McCreight, E.M.: Organization and maintenance of large ordered indexes. Acta Inform. **1**, 173–189 (1972)

Bayer, R., Schkolnick, M.: Concurrency of operations on B-trees. Acta Inform. **9**, 1–21 (1977)

Bazzi, R.A., Neiger, G.: Simplifying Fault-tolerance: Providing the Abstraction of Crash Failures. J. ACM **48**(3), 499–554 (2001)

Beame, P., Fich, F.E.: Optimal bounds for the predecessor problem and related problems. J. Comput. Syst. Sci. **65**(1), 38–72 (2002). See also STOC'99

Beame, P., Saks, M., Sun, X., Vee, E.: Time-space trade-off lower bounds for randomized computation of decision problems. J. ACM **50**(2), 154–195 (2003)

Beasley, J.E.: Operations research library. http://people.brunel.ac.uk/~mastjjb/jeb/info.html. Accessed 2008

Becchetti, L.: Modeling locality: A probabilistic analysis of LRU and FWF. In: Proceeding 12th European Symposium on Algorithms (ESA) (2004)

Becchetti, L., Könemann, J., Leonardi, S., Pál, M.: Sharing the cost more efficiently: improved approximation for multicommodity rent-or-buy. In: Proc. of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms, Society for Industrial and Applied Mathematics, Philadelphia, pp. 375–384 (2005)

Becchetti, L., Leonardi, S.: Nonclairvoyant scheduling to minimize the total flow time on single and parallel machines. J. ACM **51**(4), 517–539 (2004)

Becker, A., Bar-Yehuda, R., Geiger, D.: Randomized algorithms for the Loop Cutset problem. J. Artif. Intell. Res. **12**, 219–234 (2000)

Becker, A., Geiger, D.: Approximation algorithms for the Loop Cutset problem. In: Proc. 10th Conference on Uncertainty in Artificial Intelligence, pp. 60–68. Morgan Kaufman, San Fransisco (1994)

Becker, B., Gschwind, S., Ohler, T., Seeger, B., Widmayer, P.: An asymptotically optimal multiversion B-tree. VLDB J. **5**, 264–275 (1996)

Beckmann, N., Kriegel, H.-P., Schneider, R., Seeger, B.: The R*-Tree: An efficient and robust access method for points and rectangles. In: *Proceedings of the ACM International Conference on Management of Data*, Atlantic City, New Jersey, pp. 322–331. ACM Press, New York (1990)

Bedathur, S.J., Haritsa, J.R.: Engineering a fast online persistent suffix tree construction., In: Proc. 20th International Conference on Data Engineering, pp. 720–731, Boston, USA (2004)

Beigel, R., Alon, N., Apaydin, M.S., Fortnow, L., Kasif, S.: An optimal procedure for gap closing in whole genome shotgun sequencing. Proc. RECOMB, ACM Press pp. 22–30. (2001)

Beimel, A., Bergadano, F., Bshouty, N.H., Kushilevitz, E., Varricchio, S.: Learning Functions Represented as Multiplicity Automata. J. ACM **47**, 506–530 (2000)

Beimel, A., Bergadano, F., Bshouty, N.H., Kushilevitz, E., Varricchio, S.: On the applications of multiplicity automata in learning. In: Proc. of the 37th Annu. IEEE Symp. on Foundations of Computer Science, pp. 349–358, IEEE Comput. Soc. Press, Los Alamitos (1996)

Beimel, A., Kushilevitz, E.: Learning boxes in high dimension. In: Ben-David S. (ed.) 3rd European Conf. on Computational Learning Theory (EuroCOLT '97), Lecture Notes in Artificial Intelligence, vol. 1208, pp. 3–15. Springer, Berlin (1997) Journal version: Algorithmica **22**, 76–90 (1998)

Bein, W., Chrobak, M., Larmore, L.L.: The 3-server problem in the plane. Theor. Comput. Sci. **287**, 387–391 (2002)

Beirouti, R., Snoeyink, J.: Implementations of the LMT Heuristic for Minimum Weight Triangulation. Symposium on Computational Geometry, pp. 96–105, Minneapolis, Minnesota, June 7–10, 1998

Belady, L.A.: A study of replacement algorithms for virtual storage computers. IBM Syst. J. **5**, 78–101 (1966)

Bell, T.C., Cleary, J.G., Witten, I.H.: Text compression. Prentice Hall, NJ (1990)

Bellare, M., Coppersmith, D., Håstad, J., Kiwi, M., Sudan, M.: Linearity testing over characteristic two. IEEE Trans. Inf. Theory **42**(6), 1781–1795 (1996)

Ben-David, S., Borodin, A., Karp, R.M., Tardos, G., Wigderson, A.: On the power of randomization in on-line algorithms. Algorithmica **11**, 2–14 (1994)

Ben-David, S., Eiron, N., Long, P. M.: On the difficulty of approximately maximizing agreements. In: Proceedings of COLT, pp. 266–274 (2000)

Ben-David, S., Eiron, N., Long, P.: On the difficulty of approximately maximizing agreements. J. CSS **66**, 496–514 (2003)

Ben-Dor, A., Halevi, S., Schuster, A.: Potential function analysis of greedy hot-potato routing. Theor. Comput. Syst. **31**(1), 41–61 (1998)

Ben-Dor, A., Lancia, G., Perone, J., Ravi, R.: Banishing bias from consensus sequences. In: Proc. 8th Ann. Combinatorial Pattern Matching Conf., pp. 247–261. (1997)

Ben-Or, M.: Another advantage of free choice (extended abstract): Completely asynchronous agreement protocols. In: PODC '83: Proceedings of the second annual ACM symposium on Principles of distributed computing, pp. 27–30. ACM Press, New York (1983)

Ben-Or, M.: Another advantage of free choice: Completely asynchronous agreement protocols. In: Proc. 22nd Annual ACM Symposium on the Principles of Distributed Computing, 1983, pp. 27–30

Ben-Or, M., Coppersmith, D., Luby, M., Rubinfeld, R.: Non-abelian homomorphism testing, and distributions close to their self-convolutions. In: Proceedings of APPROX-RANDOM. Lecture Notes in Computer Science, vol. 3122, pp. 273–285. Springer, Berlin Heidelberg (2004)

Ben-Or, M., Crépeau, C., Gottesman, D., Hassidim, A., Smith, A.: Secure multiparty quantum computation with (only) a strict honest majority. In: Proceedings of the 47th Symposium on Foundations of Computer Science (FOCS'06), 2006, pp. 249–260

Ben-Or, M., El-Yaniv, R.: Optimally-resilient interactive consistency in constant time. Distrib. Comput. **16**(4), 249–262 (2003)

Ben-Or, M., Horodecki, M., Leung, D.W., Mayers, D., Oppenheim, J.: The universal composable security of quantum key distribution. In: Second Theory of Cryptography Conference TCC. Lecture Notes in Computer Science, vol. 3378, pp. 386–406. Springer, Berlin (2005). Also available at http://arxiv.org/abs/quant-ph/0409078

Ben-Sasson, E., Sudan, M., Vadhan, S., Wigderson, A.: Randomness-efficient low degree tests and short pcps via epsilon-biased sets. In: Proceedings of the Thirty-Fifth Annual ACM Symposium on the Theory of Computing, pp. 612–621. ACM, New York (2003)

Benczúr, A.A.: Counterexamples for Directed and Node Capacitated Cut-Trees. SIAM J. Comput. **24**(3), 505–510 (1995)

Benczúr, A.A., Karger, D.R.: Approximating s-t minimum cuts in $\tilde{O}(n^2)$ time. In: STOC '96: Proceedings of the twenty-eighth annual ACM symposium on Theory of computing, pp. 47–55. ACM Press, New York (1996)

Bender, M., Cole, R., Demaine, E., Farach-Colton, M.: Scanning and traversing: Maintaining data for traversals in a memory hierarchy. In: Proc. 10th Annual European Symposium on Algorithms. LNCS, vol. 2461, pp. 139–151. Springer, Berlin (2002)

Bender, M., Cole, R., Raman, R.: Exponential structures for cache-oblivious algorithms. In: Proc. 29th International Colloquium on Automata, Languages, and Programming. LNCS, vol. 2380, pp. 195–207. Springer, Berlin (2002)

Bender, M., Demaine, E., Farach-Colton, M.: Efficient tree layout in a multilevel memory hierarchy. In: Proc. 10th Annual European Symposium on Algorithms. LNCS, vol. 2461, pp. 165–173. Springer, Berlin (2002). Full version at http://arxiv.org/abs/cs/0211010

Bender, M.A., Brodal, G.S., Fagerberg, R., Ge, D., He, S., Hu, H., Iacono, J., López-Ortiz, A.: The cost of cache-oblivious searching. In: Proc. 44th Annual IEEE Symposium on Foundations of Computer Science, pp. 271–282. IEEE Computer Society Press, Los Alamitos (2003)

Bender, M.A., Demaine, E.D., Farach-Colton, M.: Cache-oblivious B-trees. SIAM J. Comput. **35**(2), 341–358 (2005). Conference version appeared at FOCS (2000)

Bender, M.A., Demaine, E.D., Farach-Colton, M.: Cache-oblivious *B*-trees. In: 41st Annual Symposium on Foundations of Computer Science, pp. 399–409. IEEE Computer Society Press, Los Alamitos (2000)

Bender, M.A., Duan, Z., Iacono, J., Wu, J.: A locality-preserving cache-oblivious dynamic dictionary. J. Algorithms **53**(2), 115–136 (2004). Conference version appeared at SODA (2002)

Bender, M.A., Farach-Colton, M.: The LCA problem revisited. In: Proceedings of the 4th Latin American Symposium on Theoretical Informatics. Lecture Notes in Computer Science, vol. 1776, Berlin, pp. 88–94. Springer, London (2000)

Bender, M.A., Farach-Colton, M., Fineman, J.T., Fogel, Y.R., Kuszmaul, B.C., Nelson, J.: Cache-oblivious streaming B-trees. In: Proc. 19th Annual ACM Symposium on Parallel Algorithms and Architectures, pp. 81–92. ACM, New York (2007)

Bender, M.A., Farach-Colton, M., Kuszmaul, B.C.: Cache-oblivious string B-trees. In: Proc. 25th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, pp. 233–242. ACM, New York (2006)

Bender, M.A., Fernandez, A., Ron, D., Sahai, A., Vadhan, S.: The power of a pebble: Exploring and mapping directed graphs. In: Proc. 30th Ann. Symp. on Theory of Computing, pp. 269–278. Dallas, 23–26 May 1998

Bender, M.A., Fineman, J.T., Gilbert, S., Kuszmaul, B.C.: Concurrent cache-oblivious B-trees. In: Proc. 17th Annual ACM Symposium on Parallel Algorithms, pp. 228–237. ACM, New York (2005)

Benioff, P.: Space searches with a quantum robot. In: Quantum computation and information (Washington, DC, 2000). Contemp. Math., vol. 305, pp. 1–12. Amer. Math. Soc. Providence, RI (2002)

Benner, S.A., Cohen, M.A., Gonnet, G.H.: Empirical and structural models for insertions and deletions in the divergent evolution of proteins. J. Mol. Biol. **229**, 1065–1082 (1993)

Bennett, C.H., Bernstein, E., Brassard, G., Vazirani, U.: Strengths and weaknesses of quantum computing. SIAM J. Comput. **26**(5), 1510–1523 (1997)

Bennett, C.H., Brassard, G.: Quantum cryptography: Public-key distribution and coin tossing. In: Proceedings of IEEE International Conference on Computers, Systems and Signal Processing, pp. 175–179. IEEE Computer Society Press, Los Alamitos (1984)

Bennett, C.H., Brassard, G., Crepeau, C., Jozsa, R., Peres, A., Wootters, W.K.: Teleporting an unknown quantum state via dual classical and Einstein-Podolsky-Rosen channels. Phys. Rev. Lett. **70**, 1895–1899 (1993)

Bennett, C.H., Brassard, G., Crépeau, C., Maurer, U.: Generalized privacy amplification. IEEE Trans. Inf. Theory **41**(6), 1915–1923 (1995)

Bennett, C.H., Brassard, G., Popescu, S., Schumacher, B., Smolin, J., Wootters, W.: Purification of noisy entanglement and faithful teleportation via noisy channels. Phys. Rev. Lett. **76**, 722–726 (1996)

Bennett, C.H., DiVincenzo, D.P., Smolin, J.A., Terhal, B.M., Wootters, W.K.: Remote state preparation. Phys. Rev. Lett. **87**, 077902 (2001)

Bennett, C.H., DiVincenzo, D.P., Smolin, J.A., Wootters, W.K.: Mixed-state entanglement and quantum error correction. Phys. Rev. A **54**, 3824–3851 (1996)

Bennett, C.H., Hayden, P., Leung, W., Shor, P.W., Winter, A.: Remote preparation of quantum states. IEEE Trans. Inform. Theory **51**, 56–74 (2005)

Bennett, C.H., Li, M., Ma, B.: Chain letters and evolutionary histories. Sci. Am. **288**, 76–81 (2003)

Bennett, C.H., Wiesner, S.J.: Communication via one- and two-particle operators on Einstein-Podolsky-Rosen states. Phys. Rev. Lett. **69**, 2881–2884 (1992)

Benoit, D., Demaine, E., Munro, J.I., Raman, R., Raman, V., Rao, S.S.: Representing trees of higher degree. Algorithmica **43**, 275–292 (2005)

Benson, G.: Tandem Repeats Finder: a program to analyze DNA sequences. Nucleic Acids Res. **27**, 573–580 (1999)

Bent, S.W., Sleator, D.D., Tarjan, R.E.: Biased search trees. SIAM J. Comput. **14**(3), 545–568 (1985)

Benteley, J.L., McGeoch, C.C.: Amortized analyses of self-organizing sequential search heuristics. Commun. ACM **28**, 404–411 (1985)

Bentley, J.: Programming Pearls. Addison-Wesley, Reading (1986)

Bentley, J., Sedgewick, R.: Algorithm alley: Sorting strings with three-way radix quicksort. Dr. Dobb's J. Softw. Tools **23**, 133–134, 136–138 (1998)

Bentley, J.L., Johnson, D.S., Leighton, F.T., McGeoch, C.C.: An experimental study of bin packing. In: Proc. of the 21st Annual Allerton Conference on Communication, Control, and Computing, Urbana, University of Illinois, 1983 pp. 51–60

Bentley, J.L., Johnson, D.S., Leighton, F.T., McGeoch, C.C., McGeoch, L.A.: Some unexpected expected behavior results for bin packing. In: Proc. of the 16th Annual ACM Symposium on Theory of Computing, pp. 279–288. ACM, New York (1984)

Bentley, J.L., McIlroy, M.D.: Engineering a sort function. Softw. Pract. Exp. **23**, 1249–1265 (1993)

Bentley, J.L., Sleator, D.S., Tarjan, R.E., Wei, V.K.: A locally adaptive data compression scheme. Commun. ACM **29**, 320–330 (1986)

Berberich, E., Eigenwillig, A., Hemmer, M., Hert, S., Schmer, K. M., Schmer, E.: A computational basis for conic arcs and boolean operations on conic polygons. In: 10th European Symposium on Algorithms (ESA'02), pp. 174–186, (2002) Lecture Notes in CS, No. 2461

Bergadano, F., Catalano, D., Varricchio, S.: Learning sat-*k*-DNF formulas from membership queries. In: Proc. of the 28th Annu. ACM Symp. on the Theory of Computing, pp. 126–130. ACM Press, New York (1996)

Bergadano, F., Varricchio, S.: Learning behaviors of automata from multiplicity and equivalence queries. In: Proc. of 2nd Italian Conf. on Algorithms and Complexity. Lecture Notes in Computer Science, vol. 778, pp. 54–62. Springer, Berlin (1994). Journal version: SIAM J. Comput. **25**(6), 1268–1280 (1996)

Bergadano, F., Varricchio, S.: Learning behaviors of automata from shortest counterexamples. In: EuroCOLT '95, Lecture Notes in Artificial Intelligence, vol. 904, pp. 380–391. Springer, Berlin (1996)

Berger, A., Czumaj, A., Grigni, M., Zhao, H.: Approximation schemes for minimum 2-connected spanning subgraphs in weighted planar graphs. Proc. 13th Annual European Symposium on Algorithms, pp. 472–483. (2005)

Bergeron, A., Chauve, C., Hartman, T., St-Onge, K.: On the properties of sequences of reversals that sort a signed permutation. Proceedings of JOBIM'02, 99–108 (2002)

Bergeron, A., Mixtacki, J., Stoye, J.: The inversion distance problem. In: Gascuel, O. (ed.) Mathematics of evolution and phylogeny. Oxford University Press, USA (2005)

Bergeron, A., Stoye, J.: On the similarity of sets of permutations and its applications to genome comparison. J. Comput. Biol. **13**(7), 1340–1354 (2006)

Bergkvist, A., Damaschke, P.: Fast algorithms for finding disjoint subsequences with extremal densities. In: Proceedings of the 16th Annual International Symposium on Algorithms and Computation. LNCS, vol. 3827, pp. 714–723 (2005)

Berkhin, P.: A survey on PageRank computing. Internet Math. **2**(1), 73–120 (2005)

Berlekamp, E., McEliece, R., van Tilborg, H.: On the inherent intractability of certain coding problems. IEEE Trans. Inf. Theory **24**, 384–386 (1978)

Berman, C.L.: Ordered Binary Decision Diagrams and Circuit Structure. In: IEEE International Conference on Computer Design. (1989)

Berman, P., Blum, A., Fiat, A., Karloff, H., Rosén, A., Saks, M.: Randomized robot navigation algorithms. In: Proceedings, Seventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 75–84 (1996)

Berman, P., Charikar, M., Karpinski, M.: On-line load balancing for related machines. J. Algorithms **35**, 108–121 (2000)

Berman, P., Coulston, C.: On-line algorithms for Steiner tree problems. In: Proc. of the 29th Annual ACM Symposium on Theory of Computing, pp. 344–353. Association for Computing Machinery, New York (1997)

Berman, P., Garay, J.A., Perry, K.J.: Bit Optimal Distributed Consensus. In: Yaeza-Bates, R., Manber, U. (eds.) Computer Science Research, pp. 313–322. Plenum Publishing Corporation, New York (1992)

Berman, P., Garay, J.A., Perry, K.J.: Optimal Early Stopping in Distributed Consensus. In: Proc. 6th International Workshop on Distributed Algorithms (WDAG), pp. 221–237, Israel, November 1992

Berman, P., Hannenhalli, S.: Fast sorting by reversal. In: Hirschberg, D.S., Myers, E.W. (eds.) Proc. 7th Ann. Symp. Combinatorial Pattern Matching (CPM96). Lecture Notes in Computer Science, vol. 1075, pp. 168–185. Laguna Beach, CA, June 1996. Springer (1996)

Berman, P., Hannenhalli, S.: Fast Sorting by Reversal, proceedings of CPM '96. Lecture notes in computer science **1075**, 168–185 (1996)

Berman, P., Karpinski, M.: Approximability of hypergraph minimum bisection. ECCC Report TR03-056, Electronic Colloquium on Computational Complexity, vol. 10 (2003)

Berman, P., Karpinski, M., Larmore, L., Plandowski, W., Rytter, W.: On the complexity of pattern matching for highly compressed two dimensional texts. Proceeding of 8th Annual Symposium on Combinatorial Pattern Matching (CPM 97). LNCS, vol. 1264, pp. 40–51. Springer, Berlin (1997)

Berman, P., Ramaiyer, V.: Improved approximations for the Steiner tree problem. J. Algorithms **17**, 381–408 (1994)

Bern, M., Eppstein, D.: Approximation algorithms for geometric problems. In: Hochbaum, D. (ed.) Approximation Algorithms for NP-hard problems. PWS Publishing, Boston (1996)

Bern, M., Plassmann, P.: The Steiner problem with edge lengths 1 and 2. Inf. Process. Lett. **32**(4), 171–176 (1989)

Bernhart, F., Kainen P.C.: The book thickness of a graph. J. Comb. Theory B **27**(3), 320–331 (1979)

Bernhart, S., Hofacker, I.L., Stadler, P.: Local RNA base pairing probabilities in large sequences. Bioinformatics **22**, 614–615 (2006)

Bernhart, S.H., Tafer, H., Mückstein, U., Flamm, C., Stadler, P.F., Hofacker, I.L.: Partition function and base pairing probabilities of RNA heterodimers. Algorithms Mol. Biol. **1**, 3 (2006)

Bernstein, E., Vazirani, U.: Quantum complexity theory. SIAM J. Comput. **26**(5), 1411–1473 (1997)

Berrou, C., Glavieux, A., Thitimajshima, P.: Near Shannon limit error-correcting coding and decoding: turbo-codes. In: Proc. IEEE Int. Conf. Comm. (ICC), pp. 1064–1070. Geneva, 23–26 May 1993

Berry, V., Guillemot, S., Nicolas, F., Paul, C.: On the approximation of computing evolutionary trees. In: Wang, L. (ed.) Proc. of the 11th Annual International Conference on Computing and Combinatorics (COCOON'05). LNCS, vol. 3595, pp. 115–125. Springer, Berlin (2005)

Berry, V., Nicolas, F.: Improved parameterized complexity of the maximum agreement subtree and maximum compatible tree problems. IEEE/ACM Trans. Comput. Biology Bioinform. **3**(3), 289–302 (2006)

Berry, V., Nicolas, F.: Maximum agreement and compatible supertrees. J. Discret. Algorithms (2006)

Berry, V., Nicolas, F.: Maximum agreement and compatible supertrees. J. Discret. Algorithms. Algorithmica, Springer, New York (2008)

Berry, V., Peng, Z.S., Ting, H.-F.: From constrained to unconstrained maximum agreement subtree in linear time. Algorithmica, to appear (2006)

Bertier, M., Marin, O., Sens, P.: Performance analysis of a hierarchical failure detector. In: International Conference on Dependable Systems and Networks (DSN 2003), San Francisco, CA, USA, Proceedings, pp. 635–644. 22–25 June 2003

Bertsekas, D.P., Gallager, R.G.: Data Networks, 2nd edn. Prentice Hall, Englewood Cliffs (1992)

Bertsekas, D.P., Tsitsiklis, J. N.: Neuro-Dynamic Programming. Athena Scientific, Belmont (1996)

Bertsimas, D., Niño-Mora, J.: Conservation laws, extended polymatroids and multiarmed bandit problems: polyhedral approaches to indexable systems. Math. Oper. Res. **21**(2), 257–306 (1996)

Besmaphyatnikh, S., Segal, M.: Enumerating longest increasing subsequences and patience sorting. Inform. Proc. Lett. **76**(1–2), 7–11 (2000)

Bespamyatnikh, S.: An Optimal Algorithm for Closest-Pair Maintenance. Discret. Comput. Geom. **19**(2), 175–195 (1998)

Bespamyatnikh, S.: On Constructing Minimum Spanning Trees in R_1^k. Algorithmica **18**(4), 524–529 (1997)

Bhalgat, A., Hariharan, R., Kavitha, T., Panigrahi, D.: An $\tilde{O}(mn)$ Gomory-Hu tree construction algorithm for unweighted graphs. In: Proc. of the 39th Annual ACM Symposium on Theory of Computing, San Diego 2007

Bhatt, S.N., Leighton, F.T.: A framework for solving vlsi graph layout problems. J. Comput. Syst. Sci. **28**(2), 300–343 (1984)

Biere, A., Cimatti, A., Clarke, E., Fujita, M., Zhu, Y.: Symbolic Model Checking Using Sat Procedures Instead of BDDs. In: ACM Design Automation Conference. (1999)

Bikhchandani, S., Chatterjee, S., Lavi, R., Mu'alem, A., Nisan, N., Sen, A.: Weak monotonicity characterizes deterministic dominant-strategy implementation. Econometrica **74**, 1109–1132 (2006)

Billoud, B., Kontic, M., Viari, A.: Palingol a declarative programming language to describe nucleic acids' secondary structures and to scan sequence database. Nucleic. Acids. Res. **24**, 1395–1403 (1996)

Bini, E., Buttazzo, G., Buttazzo, G.: Rate monotonic scheduling: The hyperbolic bound. IEEE Trans. Comput. **52**, 933–942 (2003)

Bininda-Emonds, O., Gittleman, J., Steel, M.: The (super)tree of life: Procedures, problems, and prospects. Ann. Rev. Ecol. System. **33**, 265–289 (2002)

Biran, O., Moran, S., Zaks, S.: A combinatorial characterization of the distributed 1-solvable tasks. J. Algorithms **11**(3), 420–440 (1990)

Bird, R.S.: Two dimensional pattern matching. Inf. Process. Lett. **6**, 168–170 (1977)

Birman, K.: Building Secure and Reliable Network Applications. Manning, (1996)

Biró, P., Cechlárová, K.: Inapproximability of the kidney exchange problem. Inf. Proc. Lett. **101**(5), 199–202 (2007)

Bisht, L., Bshouty, N.H., Mazzawi, H.: On Optimal Learning Algorithms for Multiplicity Automata. In: Proc. of 19th Annu. ACM Conf. Comput. Learning Theory, Lecture Notes in Computer Science. vol. 4005, pp. 184–198. Springer, Berlin (2006)

Bixby, R.E., Wagner, D.K.: An almost linear-time algorithm for graph realization. Math. Oper. Res. **13**, 99–123 (1988)

Björklund, A., Husfeldt, T.: Exact algorithms for exact satisfiability and number of perfect matchings. In: Proc. 33rd ICALP. LNCS, vol. 4051, pp. 548–1559. Springer (2006). Algorithmica, doi:10.1007/s00453-007-9149-8

Björklund, A., Husfeldt, T.: Finding a path of superlogarithmic length. SIAM J. Comput. **32**(6), 1395–1402 (2003)

Björklund, A., Husfeldt, T., Kaski, P., Koivisto, M.: Fourier meets Möbius: fast subset convolution. In: Proceedings of the 39th Annual ACM Symposium on Theory of Computing (STOC), San Diego, CA, June 11–13, 2007. Association for Computing Machinery, pp. 67–74. New York (2007)

Björklund, A., Husfeldt, T., Koivisto, M.: Set partitioning via inclusion–exclusion. SIAM J. Comput.

Blanchette, M.: Algorithms for phylogenetic footprinting. In: RECOMB01: Proceedings of the Fifth Annual International Conference on Computational Molecular Biology, pp. 49–58. ACM Press, Montreal (2001)

Blanchette, M.: Algorithms for phylogenetic footprinting. Ph. D. thesis, University of Washington (2002)

Blanchette, M., Schwikowski, B., Tompa, M.: Algorithms for phylogenetic footprinting. J. Comput. Biol. **9**(2), 211–223 (2002)

Blanchette, M., Tompa, M.: Discovery of regulatory elements by a computational method for phylogenetic footprinting. Genome Res. **12**, 739–748 (2002)

Blanchette, M., Tompa, M.: Footprinter: A program designed for phylogenetic footprinting. Nucleic Acids Res. **31**(13), 3840–3842 (2003)

Blelloch, G.E., Leiserson, C.E., Maggs, B.M., Plaxton, C.G., Smith, S.J., Zagha, M.: An experimental analysis of parallel sorting algorithms. Theor. Comput. Syst. **31**(2), 135–167 (1998)

Blin, G., Fertin, G., Vialette, S.: Extracting 2-intervals subsets from 2-interval sets. Theor. Comput. Sci. **385**(1–3), 241–263 (2007)

Blin, G., Fertin, G., Vialette, S.: New results for the 2-interval pattern problem. In: Proc. 15th Annual Symposium on Combinatorial Pattern Matching (CPM). Lecture Notes in Computer Science, vol. 3109. Springer, Berlin (2004)

Block., H. D.: The perceptron: A model for brain functioning. Rev. Mod. Phys. **34**, 123–135 (1962)

Bloom, B.: Constructing two-writer atomic registers. IEEE Trans. Comput. **37**(12), 1506–1514 (1988)

Blum, A.: Learning a function of r relevant variables (open problem). In: Proceedings of the 16th Annual Conference on Learning Theory, pp. 731–733, Washington, 24–27 August 2003

Blum, A.: New approximations for graph coloring. J. ACM **41**(3), 470–516 (1994)

Blum, A., Chawla, S., Kalai, A.: Static optimality and dynamic search-optimality in lists and trees. Algorithmica **36**, 249–260 (2003)

Blum, A., Chawla, S., Kalai, A.: Static optimality and dynamic search-optimality in lists and trees. In: Proc. 13th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 1–8 (2002)

Blum, A., Dunagan J. D.: Smoothed analysis of the perceptron algorithm for linear programming. In: SODA, (2002)

Blum, A., Frieze, A., Kannan, R., Vempala, S.: A polynomial time algorithm for learning noisy linear threshold functions. Algorithmica **22**(1/2), 35–52 (1997)

Blum, A., Furst, M., Jackson, J., Kearns, M., Mansour, Y., Rudich, S.: Weakly learning DNF and characterizing statistical query learning using Fourier analysis. In: Proceedings of the 26th Annual ACM Symposium on Theory of Computing, pp. 253–262. Association for computing Machinery, New York (1994)

Blum, A., Hartline, J.: Near-optimal online auctions. In: Proc. of the 16th Symposium on Discrete Algorithms (SODA), 2005

Blum, A., Hellerstein, L., Littlestone, N.: Learning in the presence of finitely or infinitely many irrelevant attributes. J. Comp. Syst. Sci. **50**(1), 32–40 (1995)

Blum, A., Kalai, A., Wasserman, H.: Noise-tolerant learning, the parity problem, and the statistical query model. J. ACM **50**(4), 506–519 (2003)

Blum, A., Karger, D.: An $\tilde{O}(n^{3/14})$-coloring for 3-colorable graphs. Inf. Process. Lett. **61**(6), 49–53 (1997)

Blum, A., Khardon, R., Kushilevitz, E., Pitt, L., Roth, D.: On learning read-*k*-satisfy-*j* DNF. In: Proc. of 7th Annu. ACM Conf. on Comput. Learning Theory, pp. 110–117. ACM Press, New York (1994)

Blum, A., Konjevod, G., Ravi, R., Vempala, S.: Semi-definite relaxations for minimum bandwidth and other vertex-ordering problems. Theor. Comput. Sci. **235**(1), 25–42 (2000), Selected papers in honor of Manuel Blum (Hong Kong, 1998)

Blum, A., Li, M., Tromp, J., Yannakakis, M.: Linear approximation of shortest superstrings. J. ACM **41**, 630–47 (1994)

Blum, A., Raghavan, P., Schieber, B.: Navigating in Unfamiliar Geometric Terrain. In: On Line Algorithms, pp. 151–155, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, Providence RI (1992) Preliminary Version in STOC 1991, pp. 494–504

Blum, A., Raghavan, P., Schieber, B.: Navigating in unfamiliar geometric terrain. In: Proceedings 23rd ACM Symposium on Theory of Computing (STOC), pp. 494–504 (1991)

Blum, A., Raghavan, P., Schieber, B.: Navigating in unfamiliar geometric terrain. SIAM J. Comput. **26**(1), 110–137 (1997)

Blum, A., Sandholm, T., Zinkevich, M.: Online algorithms for market clearing. J. ACM **53**(5), 845–879 (2006)

Blum, A.L., Rivest, R.L.: Training a 3-node neural network is NP-complete. Neural Netw. **5**(1), 117–127 (1992)

Blum, J., Ding, M., Thaeler, A., Cheng, X.: Applications of Connected Dominating Sets in Wireless Networks. In: Du, D.-Z., Pardalos, P. (eds.) Handbook of Combinatorial Optimization, pp. 329–369. Kluwer Academic (2004)

Blum, L., Blum, M.: Toward a mathematical theory of inductive inference. Inform. Control **28**(2), 125–155 (1975)

Blum, M., Luby, M., Rubinfeld, R.: Self-testing/correcting with applications to numerical problems. J. CSS **47**, 549–595 (1993)

Blum, M., Micali, S.: How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits. SIAM J. Comput. **4**(13), 850–864 (1984)

Blumer, A., Ehrenfeucht, A., Haussler, D., Warmuth, M.: Learnability and the Vapnik–Chervonenkis dimension. J. ACM **36**(4), 929–965 (1989)

Blumrosen, L., Nisan, N.: On the computational power of iterative auctions. In: Proc. of the 7th ACM Conference on Electronic Commerce (EC'05), 2005

Böcker, S., Mäkinen, V.: Maximum line-pair stabbing problem and its variations. In: Proc. 21st European Workshop on Computational Geometry (EWCG'05), pp. 183–186. Technische Universität Eindhoven, The Netherlands (2005)

Bodlaender, H.L.: A linear time algorithm for finding tree-decompositions of small treewidth. SIAM J. Comput. **25**, 1305–1317 (1996)

Bodlaender, H.L.: A partial *k*-arboretum of graphs with bounded treewidth. Theor. Comp. Sci. **209**, 1–45 (1998)

Bodlaender, H.L.: A tourist guide through treewidth. Acta Cybernetica **11**, 1–23 (1993)

Bodlaender, H.L.: Discovering treewidth. In: P. Vojtáš, M. Bieliková, B. Charron-Bost (eds.) Proceedings 31st Conference on Current Trends in Theory and Practive of Computer Science, SOFSEM 2005. Lecture Notes in Computer Science, vol. 3381, pp. 1–16. Springer, Berlin (2005)

Bodlaender, H.L.: On disjoint cycles. Int. J. Found. Comp. Sci. **5**(1), 59–68 (1994)

Bodlaender, H.L.: Treewidth: Characterizations, applications, and computations. In: Fomin, F.V. (ed.) Proceedings 32nd International Workshop on Graph-Theoretic Concepts in Computer Science WG'06. Lecture Notes in Computer Science, vol. 4271, pp. 1–14. Springer, Berlin (2006)

Bodlaender, H.L.: Treewidthlib. http://www.cs.uu.nl/people/hansb/treewidthlib (2004)

Bodlaender, H.L., Fellows, M.R., Warnow, T.: Two strikes against perfect phylogeny. In: Proceedings of the 19th International Colloquium on Automata, Languages and Programming (ICALP 1992). Lecture Notes in Computer Science, vol. 623, pp. 273–283. Springer, Berlin (1992)

Bodlaender, H.L., Gilbert, J.R., Hafsteinsson, H., Kloks, T.: Approximating treewidth, pathwidth, frontsize, and shortest elimination tree. J. Algorithms **18**(2), 238–255 (1995)

Bodlaender, H.L., Kloks, T., Tan, R.B., van Leeuwen, J.: Approximations for λ-Coloring of Graphs. In: Proceedings of the 17th Annual Symposium on Theoretical Aspects of Computer Science. Lecture Notes in Computer Science, vol. 1770, pp. 395-406. Springer (2000)

Bodlaender, H.L., Thilikos, D.M.: Constructive linear time algorithms for branchwidth. In: Automata, languages and programming (Bologna, 1997). Lecture Notes in Computer Science, vol. 1256, pp. 627–637. Springer, Berlin (1997)

Bodlaender, H.L., Thilikos, D.M.: Graphs with branchwidth at most three. J. Algorithms **32**, 167–194 (1999)

Boesch, F.T.: Properties of the distance matrix of a tree. Quarterly Appl. Math. **26**, 607–609 (1968)

Boeva, V.A., Régnier, M., Makeev, V.J.: SWAN: searching for highly divergent tandem repeats in DNA sequences with the evaluation of their statistical significance. Proceedings of JOBIM 2004, Montreal, Canada, p. 40 (2004)

Bogomolnaia, A., Jackson, M.O.: The Stability of Hedonic Coalition Structures. Games. Econ. Behav. **38**(2), 201–230 (2002)

Boissonnat, J.-D., Devillers, O., Pion, S., Teillaud, M., Yvinec, M.: Triangulations in CGAL. Comput. Geom. Theor. Appl. **22**(1–3), 5-19 (2002)

Boissonnat, J.-D., Teillaud, M. (eds.) Effective Computational Geometry for Curves and Surfaces. Springer, Berlin (2006)

Boldi, P., Vigna, S.: Codes for the world-wide web. Internet Math. **2**(4), 405–427 (2005)

Bollobás, B.: A probabilistic proof of an asymptotic formula for the number of labeled regular graphs. Eur. J. Comb. **1**, 311–316 (1980)

Bollobás, B.: Random Graphs. Academic Press (1985)

Bollobás, B., Coppersmith, D., Elkin M.: Sparse distance preserves and additive spanners. In: Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 2003, pp. 414–423

Boman, E., Hendrickson, B.: On spanning tree preconditioners. Manuscript, Sandia National Lab. (2001)

Boman, E., Hendrickson, B., Vavasis, S.: Solving elliptic finite element systems in near-linear time with suppost preconditioners. Manuscript, Sandia National Lab. and Cornell, http://arXiv.org/abs/cs/0407022 Accessed 9 July 2004

Boneh, D., Lipton, R.: Quantum Cryptanalysis of Hidden Linear Functions (Extended Abstract) In: Proceedings of 15th Annual International Cryptology Conference (CRYPTO'95), pp. 424–437, Santa Barbara, 27–31 August 1995

Bonizzoni, P., Della Vedova, G., Dondi, R., Li, J.: The haplotyping problem: an overview of computational models and solutions. J. Comput. Sci. Technol. **19**(1), 1–23 (2004)

Bonomi, F., Fendick, K.: The Rate-Based Flow Control for Available Bit Rate ATM Service. IEEE/ACM Trans. Netw. **9**(2), 25–39 (1995)

Bonsma, P.: Spanning trees with many leaves: new extremal results and an improved FPT algorithm. Memorandum Department of Applied Mathematics, vol. 1793, University of Twente, Enschede (2006)

Bonsma, P., Brueggemann, T., Woeginger, G.: A faster FPT algorithm for finding spanning trees with many leaves. Proceedings of MFCS 2003. Lecture Notes in Computer Science, vol. 2747, pp. 259–268. Springer, Berlin (2003)

Boost C++ Libraries, http://www.boost.org/. Accessed February 2008

Booth, K.S., Lueker, G.S.: Testing for the consecutive ones property, interval graphs, and graph planarity using *PQ*-tree algorithms. J. Comp. Syst. Sci. **13**, pp. 335–379 (1976)

Borah, M., Owens, R.M., Irwin, M.J.: An edge-based heuristic for steiner routing. IEEE Transac. Comput. Aided Des. **13**, 1563–1568 (1994)

Bordewich, M., Freedman, M., Lovasz, L., Welsh, D.: Approximate counting and Quantum computation, Combinatorics. Prob. Comput. **14**(5–6), 737–754 (2005)

Borgelt, C., Grantson, M., Levcopoulos, C.: Fixed-Parameter Algorithms for the Minimum Weight Triangulation Problem. Technical Report LU-CS-TR:2006-238, ISSN 1650-1276 Report 158. Lund University, Lund (An extended version has been submitted to IJCGA) (2006)

Borgs, C., Chayes, J., Etesami, O., Immorlica, N., Jain, K., Mahdian, M.: Bid optimization in online advertisement auctions. In: 2nd Workshop on Sponsored Search Auctions, in conjunction with the ACM Conference on Electronic Commerce (EC-06), Ann Arbor, MI, 2006

Borgs, C., Chayes, J.T., Immorlica, N., Mahdian, M., Saberi, A.: Multi-unit auctions with budget-constrained bidders. In: ACM Conference on Electronic Commerce (EC-05), 2005, pp. 44–51

Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press, Cambridge (1998)

Borodin, A., Fischer, M., Kirkpatrick, D., Lynch, N.: A time-space tradeoff for sorting on non-oblivious machines. J. Comput. Syst. Sci. **22**, 351–364 (1981)

Borodin, A., Hopcroft, J.E.: Routing, merging and sorting on parallel models of computation. J. Comput. Syst. Sci. **30**(1), 130–145 (1985)

Borodin, A., Irani, S., Raghavan, P., Schieber B.: Competitive paging with locality of reference. J. Comput. Syst. Sci. **50**(2), 244–258 (1995)

Borodin, A., Linial, N., Saks, M.E.: An optimal on-line algorithm for metrical task systems. J. ACM **39**, 745–763 (1992)

Borowsky, E., Gafni, E.: Generalized FLP impossibility result for *t*-resilient asynchronous computations. In: Proceedings of the 1993 ACM Symposium on Theory of Computing, May 1993. pp. 206–215

Borradaile, G., Kenyon-Mathieu, C., Klein, P.N.: A polynomial-time approximation scheme for Steiner tree in planar graphs. In: Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms, 2007

Borůvka, O.: O jistém problému minimálním. Práce Moravské Přírodovědecké Společnosti **3**, 37–58 (1926) (In Czech)

Borůvka, O.: Otakar Borůvka on minimum spanning tree problem (translation of both the 1926 papers, comments, history). Disc. Math. **233**, 3–36 (2001)

Boschi, D., Branca, S., Martini, F.D., Hardy, L., Popescu, S.: Experimental realization of teleporting an unknown pure quantum state via dual classical and einstein-podolski-rosen channels. Phys. Rev. Lett. **80**, 1121–1125 (1998)

Bose, P., Brodnik, A., Carlsson, S., Demaine, E., Fleischer R., López-Ortiz, A., Morin, P., Munro, J.: Online Routing in Convex Subdivisions. In: International Symposium on Algorithms and Computation (ISAAC). LNCS, vol. 1969, pp 47–59. Springer, Berlin/New York (2000)

Bose, P., Gudmundsson, J., Smid, M.: Constructing plane spanners of bounded degree and low weight. Algorithmica **42**, 249–264 (2005)

Bose, P., Gudmundsson, J., Smid, M.: Constructing plane spanners of bounded degree and low weight. In: Proceedings of European Symposium of Algorithms, University of Rome, 17–21 September 2002

Bose, P., Maheshwari, A., Narasimhan, G., Smid, M., Zeh, N.: Approximating geometric bottleneck shortest paths. Comput. Geom.: Theory Appl. **29**, 233–249 (2004)

Bose, P., Morin, P.: Competitive online routing in geometric graphs. Theor. Comput. Sci. **324**, 273–288 (2004)

Bose, P., Morin, P.: Online Routing in Triangulations. In: Proc. 10th Int. Symposium on Algorithms and Computation (ISAAC). LNCS, vol. 1741, pp 113–122. Springer, Berlin (1999)

Bose, P., Morin, P.: Online routing in triangulations. SIAM J. Comput. **33**, 937–951 (2004)

Bose, P., Morin, P., Stojmenovic, I., Urrutia, J.: Routing with guaranteed delivery in ad hoc wireless networks. ACM/Kluwer Wireless Networks **7**(6), 609–616 (2001). 3rd int. Workshop on Discrete Algorithms and methods for mobile computing and communications, 48–55 (1999)

Bose, P., Morin, P., Stojmenovic, I., Urrutia, J.: Routing with guaranteed delivery in ad hoc wireless networks. In: Proceedings of the Third International Workshop on Discrete Algorithm and Methods for Mobility, Seattle, Washington, Aug 1999, pp. 48–55

Bose, P., Smid, M., Xu, D.: Diamond triangulations contain spanners of bounded degree. In: Proceedings of the 17th International Symposium on Algorithms and Computation. Lecture Notes in Computer Science, vol. 4288, pp. 173–182. Springer, Berlin (2006)

Boser, B., Guyon, I., Vapnik, V.: A training algorithm for optimal margin classifiers. In: Proceedings of the Fifth Annual Workshop on Computational Learning Theory, Pittsburgh (1992)

Bosse, H., Byrka, J., Markakis, E.: New Algorithms for Approximate Nash Equilibria in Bimatrix Games. In: LNCS Proceedings of the 3rd International Workshop on Internet and Network Economics (WINE 2007), San Diego, 12–14 December 2007

Boston, N., Ganesan, A., Koetter, R., Pazos, S., Vontobel, P.: Papers on pseudocodewords. HP Labs, Palo Alto. http://www.pseudocodewords.info.

Bouchitté, V., Todinca, I.: Listing all potential maximal cliques of a graph. Theor. Comput. Sci. **276**(1–2), 17–32 (2002)

Bouchitté, V., Todinca, I.: Treewidth and minimum fill-in: Grouping the minimal separators. SIAM J. Comput. **31**, 212–232 (2001)

Bourgain, J.: On Lipschitz embedding of finite metric spaces in Hilbert space. Israel J. Math. **52**(1–2), 46–52 (1985)

Bourland, J.D., Wu, Q.R.: Use of shape for automated, optimized 3D radiosurgical treatment planning. SPIE Proc. Int. Symp. on Medical Imaging, pp. 553–558 (1996)

Bouwmeester, D., Pan, J.W. , Mattle, K., Eible, M., Weinfurter, H., Zeilinger, A.: Experimental quantum teleportation. Nature **390**(6660), 575–579 (1997)

Boyer, J., Myrvold, W.: Stop minding your P's and Q's: A simplified O(n) planar embedding algorithm. In: SODA '99: Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms. Philadelphia, PA, USA, Society for Industrial and Applied Mathematics, pp. 140–146 (1999)

Boyer, M., Brassard, G., Høyer, P., Tapp A.: Tight bounds on quantum searching. Fortschr. Phys. **46**(4–5), 493–505 (1998)

Boykin, P.O., Mor, T., Roychowdhury, V., Vatan, F., Vrijen, R.: Algorithmic cooling and scalable NMR quantum computers. Proc. Natl. Acad. Sci. **99**, 3388–3393 (2002)

Boykin, P.O., Roychowdhury, V.: Optimal encryption of quantum bits. Phys. Rev. A **67**, 042317 (2003)

Brace, K., Rudell, R., Bryant, R.: Efficient Implementation of a BDD Package. In: ACM Design Automation Conference. (1990)

Bracha, G.: An $O(\log n)$ expected rounds randomized Byzantine generals protocol. J. Assoc. Comput. Mach. **34**(4), 910–920 (1987)

Brafman, R., Tennenholtz, M.: R-max – a general polynomial time algorithm for near optimal reinforcement learning. J. Mach. Learn. Res. **3**, 213–231 (2002)

Braga, M.D.V., Sagot, M.F., Scornavacca, C., Tannier, E.: The Solution Space of Sorting by Reversals. In: Proceedings of ISBRA'07. Lect. Notes Comp. Sci. **4463**, 293–304 (2007)

Brainard, W.C., Scarf, H.E.: How to compute equilibrium prices in 1891. Cowles Foundation Discussion Paper 1270, August 2000

Brakmo, L.S., Peterson, L.: TCP Vegas: End-to-end Congestion Avoidance on a Global Internet. IEEE J. Sel. Areas Commun. **13**(8), 1465–1480 (1995)

Brass, P., Pach, J.: Problems and results on geometric patterns. In: Avis, D. et al. (eds.) Graph Theory and Combinatorial Optimization, pp. 17–36. Springer Science + Business Media Inc., NY, USA (2005)

Brassard, G.: Searching a quantum phone book. Science **275**(5300), 627–628 (1997)

Brassard, G., Elias, Y., Fernandez, J.M., Gilboa, H., Jones, J.A., Mor, T., Weinstein, Y., Xiao, L.: Experimental heat-bath cooling of spins. Submitted to Proc. Natl. Acad. Sci. USA. See also quant-ph/0511156 (2005)

Brassard, G., Høyer, P.: An exact quantum polynomial-time algorithm for Simon's problem. In: Proc. 5th Israeli Symp. on Theory of Computing and Systems (ISTCS), pp. 12–23. IEEE Computer Society Press, Hoboken (1997)

Brassard, G., Høyer, P., Mosca, M., Tapp A.: Quantum Amplitude Amplification and Estimation. In: Lomonaco, S.J. (ed.) Quantum Computation & Quantum Information Science, AMS Contemporary Mathematics Series Millennium Volume, vol. 305, pp. 53–74. American Mathematical Society, Providence (2002)

Brassard, G., Høyer, P., Mosca, M., Tapp, A.: Quantum amplitude amplification and estimation. In: Quantum computation and information (Washington, DC, 2000). Contemp. Math., vol. 305, pp. 53–74. American Mathematical Society, Providence, RI (2002)

Brassard, G., Høyer, P., Mosca, M., Tapp, A.: Quantum amplitude amplification and estimation. In: Quantum Computation and Quantum Information Science. AMS Contemporary Mathematics Series, vol. 305 Contemporary Mathematics, pp. 53–74, Providence (2002)

Brassard, G., Høyer, P., Tapp, A.: Quantum Algorithm for the Collision Problem. 3rd Latin American Theoretical Informatics Symposium (LATIN'98). LNCS, vol. 1380, pp. 163–169. Springer (1998)

Brassard, G., Høyer, P., Tapp, A.: Quantum cryptanalysis of hash and claw-free functions. In: Proc. 3rd Latin American Theoretical Informatics Conference (LATIN). Lecture Notes in Computer Science, vol. 1380, pp. 163–169. Springer, New York (1998)

Brayton, R., Hachtel, G., McMullen, C., Sangiovanni-Vincentelli, A.: Logic Minimization Algorithms for VLSI Synthesis. Kluwer Academic Publishers (1984)

Brayton, R.K., Hachtel, G.D., Sangiovanni-Vincentelli, A.L.: Multilevel logic synthesis. Proc. IEEE **78**(2), 264–300 (1990)

Brayton, R.K., Rudell, R., Sangiovanni-Vincentelli, A.L.: MIS: A Multiple-Level Logic Optimization. IEEE Trans. CAD **6**(6), 1061–1081 (1987)

Briest, P., Krysta, P., Vöcking, B.: Approximation techniques for utilitarian mechanism design. In: Proc. 37th Ann. ACM. Symp. on Theory of Comput. (STOC), pp. 39–48 (2005)

Brin, S., Page, L.: The anatomy of a large-scale hypertextual Web search engine. In: Proc. 7th Int. World Wide Web Conference, pp. 107–117. Elsevier Science, Amsterdam (1998)

Brisaboa, N.R., Fariña, A., Navarro, G., Esteller, M.F.: (S, C)-dense coding: An optimized compression code for natural language text databases. In: Nascimento, M.A. (ed.) Proc. Symp. String Processing and Information Retrieval. LNCS, vol. 2857, pp. 122–136, Manaus, Brazil, October 2003

Brodal, G.S.: Cache-oblivious algorithms and data structures. In: Proc. 9th Scandinavian Workshop on Algorithm Theory. LNCS, vol. 3111, pp. 3–13. Springer, Berlin (2004)

Brodal, G.S., Fagerberg, R.: Cache oblivious distribution sweeping. In: Proc. 29th International Colloquium on Automata, Languages, and Programming. LNCS, vol. 2380, pp. 426–438. Springer, Berlin (2002)

Brodal, G.S., Fagerberg, R.: Cache-oblivious string dictionaries. In: SODA: ACM-SIAM Symposium on Discrete Algorithms, pp. 581–590. ACM Press, New York (2006)

Brodal, G.S., Fagerberg, R.: On the limits of cache-obliviousness. In: Proc. 35th Annual ACM Symposium on Theory of Computing, pp. 307–315. ACM, New York (2003)

Brodal, G.S., Fagerberg, R., Jacob, R.: Cache-oblivious search trees via binary trees of small height. In: Proc. 13th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 39–48 ACM, New York (2002)

Brodal, G.S., Fagerberg, R., Meyer, U., Zeh, N.: Cache-oblivious data structures and algorithms for undirected breadth-first search and shortest paths. In: Proc. 9th Scandinavian Workshop on Algorithm Theory. LNCS, vol. 3111, pp. 480–492. Springer, Berlin (2004)

Brodal, G.S., Fagerberg, R., Vinther, K.: Engineering a cache-oblivious sorting algorithm. ACM J. Exp. Algoritmics (Special Issue of ALENEX 2004) **12**(2.2), 23 (2007)

Brodal, G.S., Jacob, R.: Time-dependent networks as models to achieve fast exact time-table queries. In: Proceedings of the 3rd Workshop on Algorithmic Methods and Models for Optimization of Railways (ATMOS'03), 2003, [1], pp. 3–15

Brodnik, A., Carlsson, S., Fredman, M.L., Karlsson, J., Munro, J.I.: Worst case constant time priority queue. J. Syst. Softw. **78**(3), 249–256 (2005). See also SODA'01

Brodnik, A., Munro, J.I.: Membership in constant time and minimum space. In: Lecture Notes in Computer Science, vol. 855, pp. 72–81, Springer, Berlin (1994). Final version: Membership in Constant Time and Almost-Minimum Space. SIAM J. Comput. **28**(5), 1627–1640 (1999)

Brönnimann, H., Goodrich, M.T.: Almost optimal set covers in finite VC-dimension. Discret. Comput. Geom. **14**(4), 463–479 (1995)

Brown, L.G.: A survey of image registration techniques. ACM Computing Surveys **24**, 325–376 (1992)

Brown, M.: Algorithm Animation. MIT Press, Cambridge, MA (1988)

Brown, M.: Perspectives on Algorithm Animation. In: Proceedings of the ACM SIGCHI'88 Conference on Human Factors in Computing Systems. Washington, D.C., May 15–19, pp. 33–38 (1988)

Brown, M.: Zeus: a System for Algorithm Animation and Multi-View Editing. In: Proceedings of the 7th IEEE Workshop on Visual Languages. Kobe, Japan, October 8–11, pp. 4–9 (1991)

Brown, M., Grundy, W., Lin, D., Cristianini, N., Sugnet, C., Furey, T., Ares Jr., M., Haussler, D.: Knowledge-based analysis of mircoarray gene expression data using support vector machines. In: Proceedings of the National Academy of Sciences **97**(1), 262–267 (2000)

Brown, M., Wilson, C.: RNA pseudoknot modeling using intersections of stochastic context free grammars with applications to database search. In: Hunter, L., Klein, T. (eds.) Proceedings of the 1st Pacific Symposium on Biocomputing, 1996, pp. 109–125

Bruno, J., Downey, P., Frederickson, G.N.: Sequencing tasks with exponential service times to minimize the expected flow time or makespan. J. ACM **28**, 100–113 (1981)

Bruno, J.L., Coffman, E.G., Sethi, R.: Scheduling independent tasks to reduce mean finishing time. Commun. ACM **17**, 382–387 (1974)

Bruno, W.J., Socci, N.D., Halpern, A.L.: Weighted Neighbor Joining: A Likelihood-Based Approach to Distance-Based Phylogeny Reconstruction. Mol. Biol. Evol. **17**, 189–197 (2000)

Bruschi, D., Del Pinto, M.: Lower bounds for the broadcast problem in mobile radio networks. Distrib. Comput. **10**(3), 129–135 (1997)

Bryand, D.: Building trees, hunting for trees, and comparing trees: theory and methods in phylogenetic analysis. In: Ph. D. thesis, Dept. Math., University of Canterbury (1997)

Bryant, R.: Graph-based Algorithms for Boolean Function Manipulation. IEEE Trans. Comp. **C-35**, 677–691 (1986)

Bshouty, N., Eiron, N., Kushilevitz, E.: PAC learning with nasty noise. TCS **288**, 255–275 (2002)

Bshouty, N., Feldman, V.: On using extended statistical queries to avoid membership queries. J. Mach. Learn. Res. **2**, 359–395 (2002)

Bshouty, N., Hellerstein, L.: Attribute-efficient learning in query and mistake-bound models. J. Comp. Syst. Sci. **56**(3), 310–319 (1998)

Bshouty, N.H.: Exact Learning Boolean Function via the Monotone Theory. Inform. Comput. **123**, 146–153 (1995)

Bshouty, N.H.: Exact learning via the monotone theory. In: Proc. of the 34th Annu. IEEE Symp. on Foundations of Computer Science, pp. 302–311. IEEE Comput. Soc. Press, Los Alamitos (1993). Journal version: Inform. Comput. **123**(1), 146–153 (1995)

Bshouty, N.H.: Simple learning algorithms using divide and conquer. In: Proc. of 8th Annu. ACM Conf. on Comput. Learning Theory, pp. 447–453. ACM Press, New York (1995). Journal version: Computational Complexity, **6**, 174–194 (1997)

Bshouty, N.H., Cleve, R., Gavaldà, R., Kannan, S., Tamon, C.: Oracles and Queries That Are Sufficient for Exact Learning. J. Comput. Syst. Sci. **52**(3), 421–433 (1996)

Bshouty, N.H., Jackson, J.C.: Learning DNF over the uniform distribution using a quantum example oracle. SIAM J. Comput. **28**, 1136–1153 (1999)

Bshouty, N.H., Jackson, J.C., Tamon, C.: More efficient PAC-learning of DNF with membership queries under the uniform distribution. J. Comput. Syst. Sci. **68**, 205–234 (2004)

Bshouty, N.H., Mossel, E., O'Donnell, R., Servedio, R.A.: Learning DNF from random walks. J. Comput. Syst. Sci. **71**, 250–265 (2005)

Bshouty, N.H., Tamon, C., Wilson, D.K.: Learning Matrix Functions over Rings. Algorithmica **22**(1/2), 91–111 (1998)

Bu, T.-M., Deng, X., Qi, Q.: Dynamics of strategic manipulation in ad-words auction. In: 3rd Workshop on Sponsored Search Auctions, in conjunction with WWW2007, Banff, Canada, 2007

Bu, T.-M., Qi, Q., Sun, A.W.: Unconditional competitive auctions with copy and budget constraints. In: Spirakis, P.G., Mavronicolas, M., Kontogiannis, S.C. (eds.) Internet and Network Economics, 2nd International Workshop, WINE 2006, Patras, Greece, 15–17 Dec 2006. Lecture Notes in Computer Science, vol. 4286, pp. 16–26. Springer, Berlin (2006)

Buchmann, J.: A subexponential algorithm for the determination of class groups and regulators of algebraic number fields. In: Goldstein, C. (ed.) Séminaire de Théorie des Nombres, Paris 1988–1989, Progress in Mathematics, vol. 91, pp. 27–41. Birkhäuser (1990)

Buchmann, J., Thiel, C., Williams, H.C.: Short representation of quadratic integers. In: Bosma, W., van der Poorten A.J. (eds.) Computational Algebra and Number Theory, Sydney 1992. Mathematics and its Applications, vol. 325, pp. 159–185. Kluwer Academic Publishers (1995)

Buchmann, J.A., Williams, H.C.: A key exchange system based on real quadratic fields (extended abstract). In: Brassard, G. (ed.)

Advances in Cryptology–CRYPTO '89. Lecture Notes in Computer Science, vol. 435, 20–24 Aug 1989, pp. 335–343. Springer (1990)

Buchsbaum, A., Kaplan, H., Rogers, A., Westbrook, J.R.: Linear-time pointer-machine algorithms for least common ancestors, MST verification and dominators. In: Proc. ACM Symp. on Theory of Computing (STOC), 1998, pp. 279–288

Buchsbaum, A.L., Caldwell, D.F., Church, K.W., Fowler, G.S., Muthukrishnan, S.: Engineering the compression of massive tables: An experimental approach. In: Proc. 11th ACM-SIAM Symp. on Discrete Algorithms, 2000, pp. 175–84

Buchsbaum, A.L., Fowler, G.S., Giancarlo, R.: Improving table compression with combinatorial optimization. J. ACM **50**, 825–851 (2003)

Buchsbaum, A.L., Goodrich, M.T., Westbrook, J.R.: Range searching over tree cross products. In: Proceedings of European Symposium on Algorithms, 2000, pp. 120–131

Buhler, J., Tompa, M.: Finding motifs using random projections. In: RECOMB01: Proceedings of the Fifth Annual International Conference on Computational Molecular Biology, 2001, pp. 69–76

Buhler, J., Tompa, M.: Finding motifs using random projections. J. Comput. Biol. **9**(2), 225–242 (2002)

Buhrman, H., Dürr, C., Heiligman, M., Høyer, P., Magniez, F., Santha, M., de Wolf, R. Quantum algorithms for element distinctness, quant-ph/0007016 (2000)

Buhrman, H., Durr, C., Heiligman, M., Høyer, P., Magniez, F., Santha, M., de Wolf, R.: Quantum algorithms for element distinctness. SIAM J. Comput. **34**(6), 1324–1330 (2005)

Buhrman, H., Dürr, C., Heiligman, M., P.Høyer, Magniez, F., Santha, M., de Wolf, R.: Quantum algorithms for element distinctness. SIAM J. Computing **34**(6), 1324–1330, (2005). Preliminary version in Proc. CCC (2001) quant-ph/0007016

Buhrman, H., Miltersen, P.B., Radhakrishnan, J., Venkatesh, S.: Are bitvectors optimal? SIAM J. Comput. **31**(6), 1723–1744 (2002)

Buhrman, H., Špalek, R.: Quantum verification of matrix products. In: Proceedings of 17th ACM-SIAM Symposium on Discrete Algorithms, pp. 880–889, Miami, FL, USA, 22–26 January 2006

Bui, T.N., Jones, C.: Finding good approximate vertex and edge partitions is NP-hard. Inform. Process. Lett. **42**(3), 153–159 (1992)

Bunch, J., Hopcroft, J.: Triangular Factorization and Inversion by Fast Matrix Multiplication. Math. Comput. **125**, 231–236 (1974)

Bunke, H., Csirik, H.: An Improved Algorithm for Computing the Edit Distance of Run Length Coded Strings. Inf. Proc. Lett. **54**, 93–96 (1995)

Burani, N., Zwicker, W.S.: Coalition formation games with separable preferences. Math. Soc. Sci. **45**, 27–52 (2003)

Burch, J.R., Clarke, E.M., McMillan, K.L., Dill, D.L.: Symbolic Model Checking: 10^{20} States and Beyond. Inf. Comp. **98**(2), 142–170 (1992)

Burd, T.D., Brodersen, R.W.: Design Issues for Dynamic Voltage Scaling, Proceedings of the 2000 international symposium on Low power electronics and design, pp. 9–14. ACM, New York, USA (2000)

Burgart, L.J., Robinson, R.A., Heller, M.J., Wilke, W.W., Iakoubova, O.K., Cheville, J.C.: Multiplex polymerase chain reaction. Mod. Pathol. **5**, 320–323 (1992)

Burkhardt, S., Kärkkäinen, J.: Fast lightweight suffix array construction and checking. In: Proc. 14th Annual Symposium on Combinatorial Pattern Matching. LNCS, vol. 2676, pp. 55–69. Springer, Berlin/Heidelberg (2003)

Burkhart, M., von Rickenbach, P., Wattenhofer, R., Zollinger, A.: Does topology control reduce interference. In: ACM Int. Symposium on Mobile Ad-Hoc Networking and Computing (MobiHoc), Tokyo, 24–26 May 2004

Burley, W.R.: Traversing layered graphs using the work function algorithm. J. Algorithms **20**, 479–511 (1996)

Burley, W.R., Irani, S.: On algorithm design for metrical task systems. Algorithmica **18**, 461–485 (1997)

Burnikel, C., Funke, S., Mehlhorn, K., Schirra, S., Schmitt, S.: A separation bound for real algebraic expressions. In: Lecture Notes in Computer Science, pp. 254–265. Springer, vol 2161 (2001)

Burns, J.E.: A formal model for message-passing systems. Indiana University, Bloomington, TR-91, USA (1980)

Burns, J.E., Lynch, N.A.: The Byzantine Firing Squad problem. Adv. Comput. Res. **4**, 147–161 (1987)

Burns, J.E., Peterson, G.L.: Constructing multi-reader atomic values from non-atomic values. In: Proc. 6th ACM Symp. Principles Distr. Comput., pp. 222–231. Vancouver, 10–12 August 1987

Burns, J.N., Lynch, N.A.: Bounds on shared-memory for mutual exclusion. Inform. Comput. **107**(2), 171–184 (1993)

Burrows, M., Wheeler, D.: A block sorting lossless data compression algorithm. Tech. Report 124, Digital Equipment Corporation (1994)

Busch, C., Herlihy, M., Wattenhofer, R.: Hard-potato routing. In: Proceedings of the 32nd Annual ACM Symposium on Theory of Computing, pp. 278–285. Portland, Oregon, United States (2000)

Busch, C., Magdon-Ismail, M., Mavronicolas, M., Spirakis, P.: Direct routing: Algorithms and Complexity. Algorithmica **45**(1), 45–68 (2006)

Busch, C., Tirthapura, S.: Analysis of link reversal routing algorithms. SIAM J. Comput. 35(2):305–326 (2005)

Busch, R., Magdon-Ismail, M., Sivrikaya, F., Yener, B.: Contention-Free MAC Protocols for Wireless Sensor Networks. In: Proc. 18th Annual Conference on Distributed Computing (DISC) (2004)

Butler, J.M.: Forensic DNA Typing: Biology and Technology Behind STR Markers. Academic Press (2001)

Butman, A., Hermelin, D., Lewenstein, M., Rawitz, D.: Optimization problems in multiple-interval graphs. In: Proc. 9th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), ACM-SIAM, 2007, pp. 268–277

Byrka, J.: An optimal bifactor approximation algorithm for the metric uncapacitated facility location problem. In: Proceedings of the 10th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX), Lecture Notes in Computer Science, vol. 4627, pp. 29–43. Springer, Berlin (2007)

Byskov, J.M.: Exact algorithms for graph colouring and exact satisfiability. Ph. D. thesis, University of Aarhus, Denmark (2004)

C. Ambühl: An optimal bound for the MST algorithm to compute energy efficient broadcast trees in wireless networks. In: Proceedings of 32th International Colloquium on Automata, Languages and Programming (ICALP). Lecture Notes in Computer Science, vol. 3580, pp. 1139–1150. Springer, Berlin (2005)

Cabello, S.: Many distances in planar graphs. In: SODA '06: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm, pp. 1213–1220. ACM Press, New York (2006)

Cai, M., Deng, X.: Approximation and computation of arbitrage in frictional foreign exchange market. Electron. Notes Theor. Comput. Sci. **78**, 1–10(2003)

Cain, J.A., Sanders, P., Wormald, N.: The random graph threshold for *k*-orientability and a fast algorithm for optimal multiple-choice allocation. In: Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '07), pp. 469–476. ACM Press, New Orleans, Louisiana, USA, 7–9 December 2007

Calabro, C., Impagliazzo, R., Kabanets, V., Paturi, R.: The Complexity of Unique *k*-SAT: An Isolation Lemma for *k*-CNFs. In: Proceedings of the Eighteenth IEEE Conference on Computational Complexity, 2003

Calabro, C., Impagliazzo, R., Paturi, R.: A duality between clause width and clause density for SAT. In: Proceedings of the 21st Annual IEEE Conference on Computational Complexity (CCC 2006), pp. 252–260. IEEE Computer Society (2006)

Calderbank, A.R., Rains, E.M., Shor, P.W., Sloane, N.J.A.: Quantum error correction via codes over GF(4). IEEE Trans. Inform. Theory **44**, 1369–1387 (1998)

Calderbank, A.R., Shor, P.W.: Good quantum error-correcting codes exist. Phys. Rev. A **54**, 1098–1105 (1996)

Caldwell, A.E., Kahng, A.B., Markov, I.L.: Optimal partitioners and end-case placers for standard-cell layout. IEEE Trans. CAD **19**(11), 1304–1314 (2000)

Calinescu, G., Karloff, H., Rabani, Y.: Approximation algorithms for the 0-extension problem. In: SODA '01: Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms, Philadelphia, PA, USA, Society for Industrial and Applied Mathematics, pp. 8–16. (2001)

Calinescu, G., Karloff, H.J., Rabani, Y.: An improved approximation algorithm for multiway cut. J. Comput. Syst. Sci. **60**(3), 564–574 (2000)

Callahan, P.: Dealing with Higher Dimensions: The Well-Separated Pair Decomposition and Its Applications. Ph. D. Thesis, The Johns Hopkins University, USA (1995)

Callahan, P.B., Kosaraju, S.R.: A decomposition of multidimensional point sets with applications to *k*-nearest-neighbors and *n*-body potential fields. J. ACM **42**, 67–90 (1995)

Callahan, P.B., Kosaraju, S.R.: Faster Algorithms for Some Geometric Graph Problems in Higher Dimensions. In: SODA 1993, pp. 291–300

Canadian Resident Matching Service (CaRMS) http://www.carms.ca/. Accessed 27 Feb 2008, JST

Canetti, R., Rabin, T.: Fast asynchronous Byzantine agreement with optimal resilience. In: Proceedings of the 25th Annual ACM Symposium on the Theory of Computing, San Diego, California, 16–18 May 1993, pp. 42–51

Cao, P., Irani, S.: Cost-aware WWW proxy caching algorithms. In: USENIX Symposium on Internet Technologies and Systems, Monterey, December 1997

Caprara, A.: Sorting by reversals is difficult. In: Proc. 1st Conf. Computational Molecular Biology (RECOMB97), pp. 75–83. ACM, Santa Fe, NM (1997)

Caprara, A.: Sorting permutations by reversals and Eulerian cycle decompositions. SIAM J. Discret. Math. **12**(1), 91–110 (1999)

Carr, R.: The Tandem global update protocol. Tandem Syst. Rev. **1**, 74–85 (1985)

Carter, J.L., Wegman, M.N.: Universal classes of hash functions. J. Comput. Syst. Sci. **18**(2), 143–154 (1979)

Cartigny, J., Ingelrest, F., Simplot-Ryl, D., Stojmenovic, I.: Localized LMST and RNG based minimum-energy broadcast protocols in ad hoc networks. Ad Hoc Netw. **3**(1), 1–16 (2004)

Cary, M., Das, A., Edelman, B., Giotis, I., Heimerl, K., Karlin, A.R., Mathieu, C., Schwarz, M.: Greedy bidding strategies for keyword auctions. In: MacKie-Mason, J.K., Parkes, D.C., Resnick, P. (eds.) Proceedings of the 8th ACM Conference on Electronic Commerce (EC-2007), San Diego, California, USA, June 11–15 2007, pp. 262–271. ACM, New York (2007)

Case, J., Smith, C.H.: Anomaly hierarchies of mechanized inductive inference. In: Proceedings of the 10th Symposium on the Theory of Computing, pp. 314–319. ACM, New York (1978)

Case, J., Smith, C.H.: Comparison of Identification Criteria for Machine Inductive Inference. Theor. Comput. Sci. **25**(2), 193–220 (1983)

Cassels, J.W.S.: An introduction to the geometry of numbers. Springer, New York (1971)

Castro, M., Druschel, P., Kermarrec, A.-M., Nandi, A., Rowstron, A., Singh, A.: Splitstream: High-bandwidth multicast in a cooperative environment. In: SOSP'03, October 2003

Castro, M., Druschel, P., Rowstron, A.: Scribe: A large-scale and decentralised application-level multicast infrastructure, IEEE J. Sel. Areas Commun. (JSAC) (Special issue on Network Support for Multicast Communications) **20**(8), 1489–1499 (2002). ISSN: 0733–8716

Cattaneo, G., Faruolo, P., Ferraro Petrillo, U., Italiano, G.F.: Maintaining Dynamic Minimum Spanning Trees: An Experimental Study. In: Proceeding 4th Workshop on Algorithm Engineering and Experiments (ALENEX 02), 6–8 Jan 2002. pp. 111–125

Cattaneo, G., Ferraro, U., Italiano, G.F., Scarano, V.: Cooperative Algorithm and Data Types Animation over the Net.J.Visual Lang.Comp. 13(4): 391– (2002)

Cechlárová, K., Dahm, M., Lacko, V.: Efficiency and stability in a discrete model of country formation. J. Glob. Opt. **20**(3–4), 239–256 (2001)

Cechlárová, K., Fleiner, T., Manlove, D.: The kidney exchange game. In: Zadnik-Stirn, L., Drobne, S. (eds.) Proc. SOR '05, pp. 77–83. Nova Gorica, September 2005

Cechlárová, K., Hajduková, J.: Computational complexity of stable partitions with $\mathcal{B}$-preferences. Int. J. Game. Theory **31**(3), 353–364 (2002)

Cechlárová, K., Hajduková, J.: Stability of partitions under WB-preferences and BW-preferences. Int. J. Inform. Techn. Decis. Mak. Special Issue on Computational Finance and Economics. **3**(4), 605–614 (2004)

Cechlárová, K., Hajduková, J.: Stability testing in coalition formation games. In: Rupnik, V., Zadnik-Stirn, L., Drobne, S. (eds.) Proceedings of SOR'99, pp. 111–116. Predvor, Slovenia (1999)

Cechlárová, K., Hajduková, J.: Stable partitions with $\mathcal{W}$-preferences. Discret. Appl. Math. **138**(3), 333–347 (2004)

Cechlárová, K., Lacko, V.: The Kidney Exchange problem: How hard is it to find a donor? IM Preprint A4/2006, Institute of Mathematics, P.J. Šafárik University, Košice, Slovakia, (2006)

Cechlárová, K., Romero-Medina, A.: Stability in coalition formation games. Int. J. Game. Theor. **29**, 487–494 (2001)

Cesa-Bianchi, N., Dichterman, E., Fischer, P., Shamir, E., Simon, H.U.: Sample-efficient strategies for learning in the presence of noise. J. ACM **46**, 684–719 (1999)

Cesa-Bianchi, N., Gentile, C.: Tracking the best hyperplane with a simple budget perceptron. In: Proceedings of the Nineteenth Annual Conference on Computational Learning Theory, (2006)

CGAL: Computational Geometry Algorithms Library, http://www.cgal.org/. Accessed February 2008

Chaitin, G.J.: Register allocation & spilling via graph coloring. In: Proceedings of the 1982 SIGPLAN Symposium on Compiler Construction (1982) pp. 98–105.

Chaitin, G.J., Auslander, M.A., Chandra, A.K., Cocke, J., Hopkins, M.E., Markstein, P.W.: Register allocation via coloring. Comp. Lang. **6**, 47–57 (1981)

Chakrabarti, A., Khot, S.: Improved lower bounds on the randomized complexity of graph properties. Proc. ICALP (2001)

Chakrabarti, A., Regev, O.: An optimal randomised cell probe lower bound for approximate nearest neighbour searching. In: Proc. 45th IEEE Symposium on Foundations of Computer Science (FOCS), 2004, pp. 473–482

Chan, C.-Y., Felber, P., Garofalakis, M., Rastogi, R.: Efficient filtering of XML documents with XPath expressions. In: *Proceedings of the 18th International Conference on Data Engineering*, San Jose, California, pp. 235–244. IEEE Computer Society, New Jersey (2002)

Chan, C.-Y., Garofalakis, M., Rastogi, R.: RE-Tree: An efficient index structure for regular expressions. In: *Proceedings of 28th International Conference on Very Large Data Bases*, Hong Kong, China, pp. 251–262. Morgan Kaufmann, Missouri (2002)

Chan, C.-Y., Garofalakis, M., Rastogi, R.: RE-Tree: An efficient index structure for regular expressions. *VLDB J.* **12**(2), 102–119 (2003)

Chan, H.-L., Lam, T.-W., Sung, W.-K., Tam, S.-L., Wong, S.-S.: A linear size index for approximate pattern matching. In: Proceedings of Symposium on Combinatorial Pattern Matching, 2006, pp. 49–59

Chan, H.-L., Lam, T.-W., Sung, W.-K., Tam, S.-L., Wong, S.-S.: Compressed indexes for approximate string matching. In: Proceedings of European Symposium on Algorithms, 2006, pp. 208–219

Chan, T.: More algorithms for all-pairs shortest paths in weighted graphs. In: Proc. 39th ACM Symposium on Theory of Computing (STOC), 2007, pp. 590–598

Chan, T., Cong, J., Sze, K.: Multilevel generalized force-directed method for circuit placement. Proc. Intl. Symp. Physical Design. ACM Press, San Francisco, 3–5 Apr 2005. pp. 185–192 (2005)

Chan, T.M.: Backward analysis of the Karger–Klein–Tarjan algorithm for minimum spanning trees. Inf. Process. Lett. **67**, 303–304 (1998)

Chan, T.M.: Euclidean bounded-degree spanning tree ratios. Discret. Comput. Geom. **32**(2), 177–194 (2004)

Chan, W.-T., Wong, P.W.H., Yung, F.C.C.: On dynamic bin packing: an improved lower bound and resource augmentation analysis. In: Proc. of the 12th Annual International Conference on Computing and Combinatorics (COCOON2006), 2006, pp. 309–319

Chandhuri, S.: More Choices Allow More Faults: Set Consensus Problems in Totally Asynchronous Systems. Inf. Comput. **105**(1), 132–158, July 1993

Chandra, T.D., Hadzilacos, V., Toueg, S.: The Weakest Failure Detector for Solving Consensus. J. ACM **43**(4), 685–722 (1996)

Chandra, T.D., Toueg, S.: Unreliable failure detectors for reliable distributed systems. J. ACM **43**(2), 225–267 (1996)

Chang, D.E., Vandersypen, L.M.K., Steffen, M.: NMR implementation of a building block for scalable quantum computation. Chem. Phys. Lett. **338**, 337–344 (2001)

Chang, E.C., Choi, S.W., Kwon, D., Park, H., Yap, C.: Shortest Paths for Disc Obstacles is Computable. In: Gao, X.S., Michelucci, D. (eds.) Special Issue on Geometric Constraints. Int. J. Comput. Geom. Appl. **16**(5–6), 567–590 (2006), Also appeared in Proc. 21st ACM Symp. Comp. Geom., pp. 116–125 (2005)

Chang, J.-M., Maxemchuk, N.F.: Reliable broadcast protocols. ACM Trans. Comput. Syst. **2**, 251–273 (1984)

Chang, J.T.: Full reconstruction of Markov models on evolutionary trees: identifiability and consistency. Math. Biosci. **137**, 51–73 (1996)

Chang, P., Mendonca, D., Yao, X., Raghavachari, M.: An evaluation of ranking methods for multiple incomplete round-robin tournaments. In: Proceedings of the 35th Annual Meeting of Decision Sciences Institute, Boston, 20–23 November 2004

Chang, S.K.: The design of network configurations with linear or piecewise linear cost functions. In: Symp. on Computer-Communications, Networks, and Teletraffic, pp. 363–369 IEEE Computer Society Press, California (1972)

Chang, S.K.: The generation of minimal trees with a Steiner topology. J. ACM **19**, 699–711 (1972)

Chang, W., Marr, T.: Approximate string matching and local similarity. In: Proc. 5th Annual Symposium on Combinatorial Pattern Matching (CPM'94). LNCS, vol. 807, pp. 259–273. Springer, Berlin, Germany (1994)

Chang, Y.-C., Chang, Y.-W., Wu, G.-M., Wu, S.-W.: B*-trees: A new representation for non-slicing floorplans. In: 37th DAC, June 2000, pp. 458–463

Chao, K.M., Miller, W.: Linear-space algorithms that build local alignments from fragments. Algorithmica **13**, 106–134 (1995)

Charikar, M., Chekuri, C., Goel, A., Guha, S.: Rounding via trees: deterministic approximation algorithms for group steiner trees and k-median. In: STOC '98: Proceedings of the thirtieth annual ACM symposium on Theory of computing, pp. 114–123. ACM Press, New York (1998)

Charikar, M., Guha, S.: Improved combinatorial algorithms for facility location problems. SIAM J. Comput. **34**(4), 803–824 (2005)

Charikar, M., Guha, S., Tardos, É., Shmoys, D.B.: A constant-factor approximation algorithm for the k-median problem (extended abstract). In: STOC '99: Proceedings of the thirty-first annual ACM symposium on Theory of computing, pp. 1–10. Atlanta, May 1-4 1999

Charikar, M., Guruswami, V., Wirth, A.: Clustering with qualitative information. In: Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS), Boston 2003, pp. 524–533

Charikar, M., Khuller, S., Mount, D., Narasimhan, G.: Facility location with outliers. In: Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 642–651. SIAM, Philadelphia (2001)

Charikar, M., Lehman, E., Liu, D., Panigraphy, R., Prabhakaran, M., Sahai, A., Shelat, A.: The smallest grammar problem. IEEE Trans. Inf. Theor. **51**, 2554–2576 (2005)

Charny, A.: An algorithm for rate-allocation in a packet-switching network with feedback. Technical Report MIT/LCS/TR-601, Massachusetts Institute of Technology, April 1994

Charras, C., Lecroq, T.: Handbook of exact string matching algorithms. King's College London Publications, London (2004)

Charron-Bost, B., Schiper A.: The "Heard-Of" model: Computing in distributed systems with benign failures. Technical Report, EPFL (2007)

Charron-Bost, B., Schiper, A.: Uniform Consensus is Harder than Consensus. J. Algorithms **51**(1), 15–37 (2004)

Chatterjee, M., Das, S., Turgut, D.: WCA: A weighted clustering algorithm for mobile ad hoc networks. J. Clust. Comput. **5**, 193–204 (2002)

Chatzigiannakis, I., Dimitriou, T., Mavronicolas, M., Nikoletseas, S., Spirakis, P.: A Comparative Study of Protocols for Efficient Data Propagation in Smart Dust Networks. In: Proc. 9th European

Symposium on Parallel Processing (EuroPar), Distinguished Paper. Lecture Notes in Computer Science, vol. 2790, pp. 1003–1016. Springer (2003) Also in the Parallel Processing Letters (PPL) Journal, Volume 13, Number 4, pp. 615–627 (2003)

Chatzigiannakis, I., Dimitriou, T., Nikoletseas, S., Spirakis, P.: A Probabilistic Algorithm for Efficient and Robust Data Propagation in Smart Dust Networks. In: Proc. 5th European Wireless Conference on Mobile and Wireless Systems (EW 2004), pp. 344–350 (2004). Also in: Ad-Hoc Netw J **4**(5), 621–635 (2006)

Chatzigiannakis, I., Dimitriou, T., Nikoletseas, S., Spirakis, P.: A probabilistic forwarding protocol for efficient data propagation in sensor networks. In: European Wireless Conference on Mobility and Wireless Systems beyond 3G (EW 2004), pp. 344–350. Barcelona, Spain, 27 February 2004

Chatzigiannakis, I., Kinalis, A., Nikoletseas, S.: An Adaptive Power Conservation Scheme for Heterogeneous Wireless Sensors. In: Proc. 17th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA 2005), ACM Press, pp. 96–105 (2005). Also in: Theory Comput Syst (TOCS) J **42**(1), 42–72 (2008)

Chatzigiannakis, I., Kinalis, A., Nikoletseas, S.: Sink Mobility Protocols for Data Collection in Wireless Sensor Networks . In: Proc. of the 4th ACM/IEEE International Workshop on Mobility Management and Wireless Access Protocols (MobiWac), ACM Press, pp. 52–59 (2006)

Chatzigiannakis, I., Kinalis, A., Nikoletseas, S.: Sink mobility protocols for data collection in wireless sensor networks. In: Zomaya, A.Y., Bononi, L. (eds.) 4th International Mobility and Wireless Access Workshop (MOBIWAC 2006), Terromolinos, pp 52–59

Chatzigiannakis, I., Mylonas, G., Nikoletseas, S.: Modeling and evaluation of the effect of obstacles on the performance of wireless sensor networks. In: 39th ACM/IEEE Simulation Symposium (ANSS), Los Alamitos, CA, USA, IEEE Computer Society, pp. 50–60 (2006)

Chatzigiannakis, I., Nikoletseas, S.: Design and analysis of an efficient communication strategy for hierarchical and highly changing ad-hoc mobile networks. J. Mobile Netw. Appl. **9**(4), 319–332 (2004). Special Issue on Parallel Processing Issues in Mobile Computing

Chatzigiannakis, I., Nikoletseas, S., Spirakis, P.: Distributed communication algorithms for ad hoc mobile networks. J. Parallel Distrib. Comput. (JPDC) **63**(1), 58–74 (2003). Special Issue on Wireless and Mobile Ad-hoc Networking and Computing, edited by Boukerche A

Chatzigiannakis, I., Nikoletseas S., Spirakis, P.: Smart dust protocols for local detection and propagation. J. Mob. Netw. (MONET) **10**, 621–635 (2005)

Chaudhuri, K., Rao, S., Riesenfeld, S., Talwar, K.: A push-relabel algorithm for approximating degree bounded MSTs. In: Proceedings of the 33rd International Colloquium on Automata, Languages and Programming (ICALP 2006), Part I. LNCS, vol. 4051, pp. 191–201. Springer, Berlin (2006)

Chaudhuri, S.: Agreement is harder than consensus: Set consensus problems in totally asynchronous systems. In: Proceedings Of The Ninth Annual ACM Symposium On Principles of Distributed Computing, August 1990. pp. 311–234

Chaudhuri, S., Herlihy, M., Lynch, N.A., Tuttle, M.R.: Tight bounds for k-set agreement. J. ACM **47**(5), 912–943 (2000)

Chaudhuri, S., Zaroliagis, C.: Shortest Paths in Digraphs of Small Treewidth. Part I: Sequential Algorithms. Algorithmca **27**(3), pp. 212–226 (2000)

Chaudhuri, S., Zaroliagis, C.: Shortest Paths in Digraphs of Small Treewidth. Part II: Optimal Parallel Algorithms. Theor. Comput. Sci. **203**(2), pp. 205–223 (1998)

Chaung, I.L., Gottesman, D.: Quantum teleportation is a universal computational primitive. Nature **402**, 390–393 (1999)

Chawla, S., Gupta, A., Räcke, H.: Embeddings of Negative-type Metrics and An Improved Approximation to Generalized Sparsest Cut. In: Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA), Vancouver, January 2005, pp. 102–111

Chawla, S., Krauthgamer, R., Kumar, R., Rabani, Y., Sivakumar, D.: On the Hardness of Approximating Sparsest Cut and Multicut. In: Proceedings of the 20th IEEE Conference on Computational Complexity (CCC), San Jose, June 2005, pp. 144–153

Chazelle, B.: A minimum spanning tree algorithm with inverse-Ackermann type complexity. J. ACM **47**(6), 1028–1047 (2000)

Che, Y.-K., Gale, I.: Standard auctions with financially constrained bidders. Rev. Econ. Stud. **65**(1), 1–21 (1998)

Cheetham, J., Dehne, F., Rau-Chaplin, A., Stege, U., Taillon, P.: Solving large FPT problems on coarse grained parallel machines. J. Comput. Syst. Sci. **67**, 691–706 (2003)

Chekuri, C., Goel, A., Khanna, S., Kumar, A.: Multi-processor scheduling to minimize flow time with epsilon resource augmentation. In: Symposium on Theory of Computing, STOC, pp. 363–372 (2004)

Chekuri, C., Hagiahayi, M.T., Kortsarz, G., Salavatipour, M.: Approximation Algorithms for Non-Uniform Buy-at-Bulk Network Design. In: Proceedings of the 47th Annual Symp. on Foundations of Computer Science, Berkeley, Oct. 2006, pp. 677–686

Chekuri, C., Khanna, S.: A PTAS for the multiple knapsack problem. In 11th ACM-SIAM Symp. on Discrete Algorithms (SODA), pp. 213–222 (2000)

Chekuri, C., Khanna, S.: Approximation algorithms for minimizing weighted completion time. In: J. Y-T. Leung (eds.) Handbook of Scheduling: Algorithms, Models, and Performance Analysis. CRC Press, Boca Raton (2004)

Chekuri, C., Khanna, S., Shepherd, F.B.: A note on multiflows and treewidth. Algorithmica, published online (2007)

Chekuri, C., Khanna, S., Shepherd, F.B.: An $O(\sqrt{n})$ approximation and integrality gap for disjoint paths and UFP. Theor. Comput. **2**, 137–146 (2006)

Chekuri, C., Khanna, S., Shepherd, F.B.: Edge Disjoint Paths in Planar Graphs. Proc. of IEEE FOCS, 2004, pp. 71–80

Chekuri, C., Khanna, S., Shepherd, F.B.: Edge-Disjoint Paths in Planar Graphs with Constant Congestion. Proc. ACM STOC, pp. 757–766 (2006)

Chekuri, C., Khanna, S., Shepherd, F.B.: Multicommodity flow, well-linked terminals, and routing problems. Proc. ACM STOC, pp. 183–192 (2005)

Chekuri, C., Khanna, S., Shepherd, F.B.: The All-or-Nothing Multicommodity Flow Problem. Proc. ACM STOC, pp. 156–165 (2004)

Chekuri, C., Motwani, R., Natarajan, B., Stein, C.: Approximation techniques for average completion time scheduling. SIAM J. Comput. **31**(1), 146–166 (2001)

Chen, B., van Vliet, A., Woeginger, G.J.: New lower and upper bounds for on-line scheduling. Oper. Res. Lett. **16**, 221–230 (1994)

Chen, C.-P., Chen, Y.-P., Wong, D.F.: Optimal wire-sizing formula under the Elmore delay model. In: Proc. ACM/IEEE Design Automation Conf., pp. 487–490 ACM, New York (1996)

Chen, C.-P., Wong, D.F.: A fast algorithm for optimal wire-sizing under Elmore delay model. In: Proc. IEEE ISCAS, vol. 4, pp. 412–415 IEEE Press, Piscataway (1996)

Chen, C.-P., Wong, D.F.: Optimal wire-sizing function with fringing capacitance consideration. In: Proc. ACM/IEEE Design Automation Conf., pp. 604–607 ACM, New York (1997)

Chen, C.P., Chu, C.N, Wong, D.F.: Fast and Exact Simultaneous Gate and Wire Sizing by Lagrangian Relax-ation. In: Proceedings of the 1998 IEEE/ACM International Conference on Computer-Aided Design, pp. 617–624. November 1998

Chen, D., Chiang, Y.J., Memon, N., Wu, X.: Optimal alphabet partitioning for semi-adaptive coding of sources of unknown sparse distributions. In: Storer, J.A., Cohn, M. (eds.) Proc. 2003 IEEE Data Compression Conference, pp. 372–381, IEEE Computer Society Press, Los Alamitos, California, March 2003

Chen, D., Cong, J., Pan, P.: FPGA design automation: a survey. Foundations and Trends in Electronic Design Automation, vol 1, no 3. Now Publishers, Hanover, USA (2006)

Chen, D.Z., Daescu, O., Klenk, K.S.: On geometric path query problems. Int. J. Comput. Geom. Appl. **11**, 617–645 (2001)

Chen, D.Z., Hu, X., Huang, Y., Li, Y., Xu, J.: Algorithms for congruent sphere packing and applications. Proc. 17th Annual ACM Symp. on Computational Geometry, pp. 212–221 (2001)

Chen, H., Frieze, A.M.: Coloring bipartite hypergraphs. In: Cunningham, H.C., McCormick, S.T., Queyranne, M. (eds.) Integer Programming and Combinatorial Optimization, 5th International IPCO Conference, Vancouver, British Columbia, Canada, June 3–5 1996. Lecture Notes in Computer Science, vol. 1084, pp. 345–358. Springer

Chen, H.Y., Kang, S.M.: icoach: A circuit optimiza-tion aid for cmos high-performance circuits. Intergr. VLSI. J. **10**(2), 185–212 (1991)

Chen, J.-H., Le, S.-Y., Maize, J.: Prediction of common secondary structures of RNAs: a genetic algorithm approach. Nucleic. Acids. Res. **28**, 991–999 (2000)

Chen, J., Fernau, H., Kanj, I.A., Xia, G.: Parametric duality and kernelization: lower bounds and upper bounds on kernel size. SIAM J. Comput. **37**(4), 1077–1106 (2007)

Chen, J., Kanj, I.A., Jia, W.: Vertex cover: further observations and further improvements. J. Algorithms **41**, 280–301 (2001)

Chen, J., Kanj, I.A., Xia, G.: Improved parameterized upper bounds for vertex cover. In: Lecture Notes in Computer Science (MFCS 2006), vol. 4162, pp. 238–249. Springer, Berlin (2006)

Chen, J., Liu, Y., Lu, S., O'Sullivan, B., Razgon, I.: A Fixed-Parameter Algorithm for the Directed Feedback Vertex Set Problem. In: 40th ACM Symposium on Theory of Computing STOC 2008, May 17–20, Victoria (BC), Canada (2008)

Chen, J., Lu, S., Sze, S., Zhang, F.: Improved algorithms for path, matching, and packing problems. Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 298–307 (2007)

Chen, J.J., Kuo, T.W., Lu, H.I.: Power-Saving Scheduling for Weakly Dynamic Voltage Scaling Devices Workshop on Algorithms and Data Structures (WADS). LNCS, vol. 3608, pp. 338–349. Springer, Berlin, Germany (2005)

Chen, K.-Y., Chao, K.-M.: On the range maximum-sum segment query problem. Proceedings of the 15th International Symposium on Algorithms And Computation. LNCS **3341**, 294–305 (2004)

Chen, K.-Y., Chao, K.-M.: Optimal algorithms for locating the longest and shortest segments satisfying a sum or an average constraint. Inf. Process. Lett. **96**, 197–201 (2005)

Chen, L., Deng, X., Fang, Q., Tian, F.: Majority equilibrium for public facility allocation. Lect. Notes Comput. Sci. **2697**, 435–444 (2002)

Chen, M.T., Seiferas, J.: Efficient and elegant subword tree construction. In: Apostolico, A., Galil, Z. (eds.) Combinatorial Algorithms on Words. Springer, New York (1985)

Chen, T., Kao, M.-Y., Tepel, M., Rush J., Church, G.: A dynamic programming approach to de novo peptide sequencing via tandem mass spectrometry. J. Comput. Biol. **8**(3), 325–337 (2001)

Chen, W., Toueg, S., Aguilera, M.K.: On the quality of service of failure detectors. IEEE Trans. Comput. **51**(1), 13–32 (2002)

Chen, X., Deng, X.: 3-Nash is PPAD-complete. ECCC, TR05-134 (2005)

Chen, X., Deng, X.: Settling the complexity of 2-player Nash-equilibrium. In: Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06). Berkeley, 21–24 October 2005

Chen, X., Deng, X., Liu, B.J.: On incentive compatible competitive selection protocol. In: COCOON'06: Proceedings of the 12th Annual International Computing and Combinatorics Conference, pp. 13–22, Taipei, 15–18 August 2006

Chen, X., Deng, X., Liu, B.J.: On incentive compatible competitive selection protocol. In: Computing and Combinatorics, 12th Annual International Conference, COCOON 2006, Taipei, Taiwan, 15 August 2006. Lecture Notes in Computer Science, vol. 4112, pp. 13–22. Springer, Berlin (2006)

Chen, X., Deng, X., Teng, S.H.: Computing Nash equilibria: approximation and smoothed complexity. In: FOCS'06: Proc. of the 47th Annual IEEE Symposium on Foundations of Computer Science, 2006, pp. 603–612

Chen, Y.H., Lu, H.I., Tang, C.Y.: Disjoint segments with maximum density. In: Proceedings of the 5th Annual International Conference on Computational Science, pp. 845–850 (2005)

Cheng, C.-H., Chen, K.-Y., Tien, W.-C., Chao, K.-M.: Improved algorithms for the *k* maximum-sum problems. Proceedings of the 16th International Symposium on Algorithms And Computation. Theoret. Comput. Sci. 362: 162–170 (2006)

Cheng, C.S., Shann, J.J.J., Chung, C.P.: Unique-order interpolative coding for fast querying and space-efficient indexing in information retrieval systems. Inf. Process. Manag. **42**(2), 407–428 (2006)

Cheng, X., Huang, X., Li, D., Wu, W., Du, D.-Z.: A polynomial-time approximation scheme for minimum connected dominating set in ad hoc wireless networks. Networks **42**, 202–208 (2003)

Cheriton, D.and Tarjan, R.E.: Finding Minimum Spanning Trees. SIAM J. Comput. **5**(4), 724–742 (1976)

Cheriyan, J., Thurimella, R.: Approximating minimum-size k-connected spanning subgraphs via matching. SIAM J. Comput. **30**(2), 528–560 (2000)

Cheriyan, J., Vempala, S., Vetta, A.: An approximation algorithm for the minimum-cost k-vertex connected subgraph. SIAM J. Comput. **32**(4), 1050–1055 (2003)

Cheriyan, J., Vetta, A.: Approximation algorithms for network design with metric costs. Proc. 37th Annual ACM Symposium on Theory of Computing, Baltimore, 22–24 May 2005, pp. 167–175. (2005)

Cherkassky, B.V., Goldberg, A.V.: Negative-Cycle Detection Algorithms. Math. Program. **85**, pp. 277–311 (1999)

Chernoff, H.: A measure of the asymptotic efficiency for tests of a hypothesis based on the sum of observations. Ann. Math. Stat. **23**, 493–509 (1952)

Cheung, C., Yu, J., Lu, H.: Constructing suffix tree for gigabyte sequences with megabyte memory. IEEE Trans. Knowl. Data Eng. **17**, 90–105 (2005)

Cheung, K., Cunningham, W.H., Tang, L.: Optimal 3-Terminal Cuts and Linear Programming. Math. Program. **105**, 389–421 (2006), Preliminary version in IPCO 1999

Cheung, K., Mosca, M.: Decomposing Finite Abelian Groups. Quantum Inf. Comp. **1**(2), 26–32 (2001)

Chew, L.P.: There are planar graphs almost as good as the complete graph. J. Comput. Syst. Sci. **39**, 205–219 (1989)

Chew, L.P.: There is a planar graph almost as good as the complete graph. In: Proceedings of the 2nd ACM Symposium on Computational Geometry, pp. 169–177 (1986)

Chew, L.P., Kedem, K.: Improvements on geometric pattern matching problems. In: Proc. Scandinavian Workshop Algorithm Theory (SWAT). LNCS, vol. 621, pp. 318–325. Springer, Berlin (1992)

Chiang, Y.-T., Lin, C.-C., Lu, H.-I.: Orderly spanning trees with applications. SIAM J. Comput. **34**(4), 924–945 (2005)

Chiaverini, J., Leibfried, D., Schaetz, T., Barrett, M.D., Blakestad, R.B., Britton, J., Itano, W.M., Jost, J.D., Knill, E., Langer, C., Ozeri, R., Wineland, D.J.: Realization of quantum error correction. Nature **432**, 602–605 (2004)

Childs, A., Cleve, R., Deotto, E., Farhi, E., Gutmann, S., Spielman, D.: Exponential algorithmic speedup by a quantum walk. In: Proc. STOC (2003)

Childs, A.M., Eisenberg, J.M.: Quantum algorithms for subset finding. Quantum Inf. Comput. **5**, 593 (2005)

Childs, A.M., Goldstone, J.: Spatial search and the Dirac equation. Phys. Rev. A. **70**, 042312 (2004)

Childs, A.M., Goldstone, J.: Spatial search by quantum walk. Phys. Rev. A **70**, 022314 (2004)

Childs, A.M., Landahl A.J., Parrilo, P.A.: Improved quantum algorithms for the ordered search problem via semidefinite programming. Phys. Rev. A **75**, 032335 (2007)

Childs, A.M., Reichardt, B.W., Špalek, R., Zhang, S.: Every NAND formula of size N can be evaluated in time $N^{1/2+o(1)}$ on a quantum computer, quant-ph/0703015 (2007)

Chinn, P.Z., Chvátalová, J., Dewdney, A.K., Gibbs, N.E.: The bandwidth problem for graphs and matrices—a survey. J. Graph Theory **6**(3), 223–254 (1982)

Chlebik M., Chlebikova J.: Approximation Hardness of the Steiner Tree Problem on Graphs. In: 8th Scandinavian Workshop on Algorithm Theory. Number 2368 in LNCS, pp. 170–179, (2002)

Chlebus, B.S., Gasieniec, L., Gibbons, A., Pelc, A., Rytter, W.: Deterministic broadcasting in ad hoc radio networks. Distrib. Comput. **15**, 27–38 (2002)

Chlebus, B.S., Kowalski, D.R.: Almost Optimal Explicit Selectors. In: Proc. 15th International Symposium on Fundamentals of Computation Theory, pp. 270–280, Lübeck, Germany (2005)

Chlebus, B.S., Kowalski, D.R.: Time and Communication Efficient Consensus for Crash Failures. In: Proc. 20th International Symposium on Distributed Computing (DISC), pp. 314–328, Sweden, September 2006

Chlebus, M., Gąsieniec, L., Östlin, A., Robson, J.M.: Deterministic broadcasting in radio networks. In: Proc. 27th International Colloquium on Automata, Languages and Programming.LNCS, vol. 1853, pp. 717–728, Geneva, Switzerland (2000)

Chockler, G., Keidar, I., Vitenberg, R.: Group communication specifications: A comprehensive study. ACM Comput. Surv. **33**, 427–469 (2001)

Choi, J., Dongarra, J.J., Pozo, R., Walker, D.W.: ScaLAPACK: A scalable linear algebra library for distributed memory concurrent computers. In: The 4th Symp. the Frontiers of Massively Parallel Computations, pp. 120–127, McLean, VA (1992)

Choi, V., Goyal, N.: An efficient approximation algorithm for point pattern matching under noise. In: Proc. 7th Latin American Symposium on Theoretical Informatics (LATIN 2006). LNCS, vol. 3882, pp. 298–310. Springer, Berlin (2006)

Chong, K., Sahni, S.: Optimal Realizations of Floorplans. In: IEEE Trans. Comput. Aided Des. **12**(6), 793–901 (1993)

Chong, K.W., Han, Y., Lam, T.W.: Concurrent Threads and Optical Parallel Minimum Spanning Trees Algorithm. J. ACM **48**(2), 297–323 (2001)

Chopra, S., Rao, M.R.: On the Multiway Cut Polyhedron. Networks **21**, 51–89 (1991)

Chor, B., Coan, B.: A simple and efficient randomized Byzantine agreement algorithm. IEEE Trans. Softw. Eng. **SE-11**(6), 531–539 (1985)

Chor, B., Dwork, C.: Randomization in Byzantine Agreement. Adv. Comput. Res. **5**, 443–497 (1989)

Chor, B., Hendy, M., Penny, D.: Analytic solutions for three-taxon ML_{MC} trees with variable rates across sites. In: Proceedings of the 1st Workshop on Algorithms in Bioinformatics (WABI 2001). Lecture Notes in Computer Science, vol. 2149, pp. 204–213. Springer (2001)

Chor, B., Moscovici, L.: Solvability in asynchronous environments. In: Proc. 30th Symposium on Foundations of Computer Science, pp. 422–427 (1989)

Chor, B., Sudan, M.: A geometric approach to betweeness. SIAM J. Discret. Math. **11**, 511–523 (1998)

Chou, P., Wu, Y., Jain, K.: Network coding for the internet. In: IEEE Communication Theory Workshop, 2004

Chowdhury, R.A., Ramachandran, V.: Cache-oblivious dynamic programming. In: Proc. 17th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 591–600. ACM-SIAM, New York (2006)

Chowdhury, R.A., Ramachandran, V.: Cache-oblivious shortest paths in graphs using buffer heap. In: Proc. 16th Annual ACM Symposium on Parallelism in Algorithms and Architectures. ACM, New York (2004)

Choy, C., Jansson, J., Sadakane, K., Sung, W.-K.: Computing the maximum agreement of phylogenetic networks. In: Proc. Computing: the 10th Australasian Theory Symposium (CATS 2004), 2004, pp. 33–45

Christie, D.A.: Genome Rearrangement Problems. Ph. D. thesis, Department of Computer Science. University of Glasgow, U.K. (1999)

Christodoulou, G., Koutsoupias, E.: On the price of anarchy and stability of correlated equilibria of linear congestion games. In: Algorithms – ESA 2005, 13th Annual European Symposium, pp. 59–70. Springer, Palma de Mallorca (2005)

Christodoulou, G., Koutsoupias, E.: The Price of Anarchy of Finite Congestion Games. In: Proc. of the 37th ACM Symp. on Th. of Comp. (STOC '05), pp. 67–73. ACM, Baltimore (2005)

Christodoulou, G., Koutsoupias, E., Nanavati, A.: Coordination mechanisms. In: Proceedings of the 31st International Colloquium on Automata, Languages and Programming (ICALP), pp. 345–357 (2004)

Christodoulou, G., Koutsoupias, E., Vidali, A.: A lower bound for scheduling mechanisms. In: Proc. 18th Symposium on Discrete Algorithms (SODA), 2007

Christofides, N.: Worst-case analysis of a new heuristic for the traveling salesman problem. In: Technical report, Graduate School of Industrial Administration. Carnegie-Mellon University, Pittsburgh (1976)

Chrobak, M.: Sigact news online algorithms column 1. ACM SIGACT News **34**, 68–77 (2003)

Chrobak, M., Gąsieniec, L., Kowalski, D.: The Wake-Up Problem in Multi-Hop Radio Networks. In: Proc. of the 15th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 992–1000 (2004)

Chrobak, M., Gąsieniec, L., Rytter, W.: A Randomized Algorithm for Gossiping in Radio Networks. In: Proc. 8th Annual International Computing Combinatorics Conference. Guilin, China, pp. 483–492 (2001) Full version in Networks **43**(2), 119–124 (2004)

Chrobak, M., Gąsieniec, L., Rytter, W.: Fast Broadcasting and Gossiping in Radio Networks,. In: Proc. 41st Annual Symposium on Foundations of Computer Science, pp. 575–581, Redondo Beach, USA (2000) Full version in J. Algorithms **43**(2) 177–189 (2002)

Chrobak, M., Karloff, H., Payne, T.H., Vishwanathan, S.: New results on server problems. SIAM J. Discret. Math. **4**(2), 172–181 (1991)

Chrobak, M., Larmore, L.L.: An optimal online algorithm for *k* servers on trees. SIAM J. Comput. **20**, 144–148 (1991)

Chrobak, M., Larmore, L.L.: Metrical service systems: Deterministic strategies. Tech. Rep. UCR-CS-93-1, Department of Computer Science, Univ. of California at Riverside (1992)

Chrobak, M., Larmore, L.L.: Metrical task systems, the server problem and the work function algorithm. In: Fiat, A., Woeginger, G.J. (eds.) Online Algorithms. The State of the Art. LNCS, vol. 1442, ch. 4, pp. 74–96. Springer, London (1998)

Chrobak, M., Sgall, J.: The weighted 2-server problem. Theor. Comput. Sci. **324**, 289–312 (2004)

Chrobak, M., Ślusarek, M.: On some packing problems relating to dynamical storage allocation. RAIRO J. Inf. Theor. Appl. **22**, 487–499 (1988)

Chu, C.: FLUTE: Fast lookup table based wirelength estimation technique. In: Proc. Intl. Conf. on Computer-Aided Design, San Jose, Nov. 2004, pp. 696–701

Chu, C., Wong, Y.C.: Fast and accurate rectilinear steiner minimal tree algorithm for vlsi design. In: International Symposium on Physical Design, pp. 28–35 (2005)

Chu, C.C.N., Wong, D.F.: A quadratic programming approach to simultaneous buffer insertion/sizing and wire sizing. IEEE Trans. Comput. Des. **18**(6), 787–798 (1999)

Chu, C.C.N., Wong, D.F.: Greedy wire-sizing is linear time. IEEE Trans. Comput. Des. **18**(4), 398–405 (1999)

Chu, Y., Rao, S.G., Zhang, H.: A case for end system multicast. In: Proceedings of ACM SIGMETRICS, Santa Clara, June 2000, pp. 1–12

Chuang, R.C.-N., Garg, A., He, X., Kao, M.-Y., Lu, H.-I.: Compact encodings of planar graphs via canonical orderings and multiple parentheses. Comput. Res. Repos. cs.DS/0102005 (2001)

Chudak, F.A., Shmoys, D.B.: Improved approximation algorithms for the uncapacitated facility location problem. SIAM J Comput. **33**(1), 1–25 (2003)

Chudak, F.A., Wiliamson, D.P.: Improved approximation algorithms for capacitated facility location problems. In: Proceedings of the 7th Conference on Integer Programing and Combinatorial Optimization (IPCO). Lecture Notes in Computer Science, vol. 1610, pp. 99–113. Springer, Berlin (1999)

Chudak, F.A., Williamson, D.P.: Improved approximation algorithms for capacitated facility location problems. Math. Program. **102**(2), 207–222 (2005)

Chun, B., Culler, D., Roscoe, T., Bavier, A., Peterson, L., Wawrzoniak, M., Bowman, M.: Planetlab: An overlay testbed for broad-coverage services. ACM SIGCOMM Comput. Commun. Rev. **33**, 3–12 (2003)

Chung, F.R.K., Hajela, D.J., Seymour, P.D.: Self-organizing sequential search and Hilbert's inequality. In: Proc. 17th Annual Symposium on the Theory of Computing pp 217–223 (1985)

Chung, F.R.K., Seymour, P.D.: Graphs with small bandwidth and cutwidth. Discret. Math. **75**(1–3), 113–119 (1989). Graph theory and combinatorics, Cambridge (1988)

Chung, K.-M., Lu, H.-I.: An optimal algorithm for the maximum-density segment problem. SIAM. J. Comput. **34**, 373–387 (2004)

Chung, R.H., Gusfield, D.: Empirical exploration of perfect phylogeny haplotyping and haplotypes. In: Proceedings of Annual International Conference on Computing and Combinatorics (COCOON). Lecture Notes in Computer Science, vol. 2697, pp. 5–9. Springer, Berlin (2003)

Chuzhoy, J., Guruswami, V., Khanna, S., Talwar, K.: Hardness of routing with congestion in directed graphs. In: STOC '07: Proceedings of the thirty-ninth annual ACM symposium on Theory of computing, pp. 165–178. ACM Press, New York (2007)

Chuzhoy, J., Khanna, S.: Polynomial flow-cut gaps and hardness of directed cut problems. In: Proceedings of the 39th ACM Symposium on Theory of Computing (STOC), San Diego, June 2007 pp. 179–188

Chuzhoy, J., Naor, J.: New Hardness Results for Congestion Minimization and Machine Scheduling. Proceedings of the 36th Annual ACM Symposium on Theory of Computing, pp. 28–34. ACM, New York (2004)

Chvátal, V.: A Greedy Heuristic for the Set-Covering Problem. Math. Oper. Res. **4**(3), 233–235 (1979)

Chvátal, V.: Tough graphs and Hamiltonian circuits. Discret. Math. **5**, 215–228 (1973)

Chvátal, V., Reed, B.: Mick gets some (the odds are on his side). In: 33rd Annual Symposium on Foundations of Computer Science, pp. 620–627. IEEE Computer Society, Pittsburgh (1992)

Cilibrasi, R., Vitanyi, P.M.B.: Clustering by compression. IEEE Trans. Inf. Theory **51**, 1523–1545 (2005)

Cimikowski, R.: Branch-and-bound techniques for the maximum planar subgraph problem. Int. J. Computer Math. **53**, 135–147 (1994)

Ciriani, V., Ferragina, P., Luccio, F., Muthukrishnan, S.: A data structure for a sequence of string acesses in external memory. ACM Trans. Algorithms **3** (2007)

Clark, B.N., Colbourn, C.J., Johnson, D.S.: Unit disk graphs. Discret. Math. **86**, 165–177 (1990)

Clark, D., Munro, J.I.: Efficient suffix trees on secondary storage. In: Proc. 7th ACM-SIAM SODA, pp. 383–391 (1996)

Clark, J., DeRose, S.: XML Path Language (XPath) Version 1.0. W3C Recommendation, http://www.w3.org./TR/xpath, Accessed Nov 1999

Clarke, E.H., Multipart pricing of public goods. Publ. Choice **2**, 19–33 (1971)

Clarkson, K.L.: Fast Expected-Time and Approximation Algorithms for Geometric Minimum Spanning Trees. In: Proc. STOC 1984, pp. 342–348

Cleary, J.G., Witten, I.H.: Data compression using adaptive coding and partial string matching. IEEE Transactions on Communications, COM–32, pp. 396–402 (1984)

Clementi, A., Crescenzi, P., Penna, P., Rossi, G., Vocca, P.: On the Complexity of Computing Minimum Energy Consumption Broadcast Subgraphs. In: Proceedings of the 18th Annual Symposium on Theoretical Aspects of Computer Science (STACS), pp. 121–131 (2001)

Clementi, A., Huiban, G., Penna, P., Rossi, G., Verhoeven, Y.: Some Recent Theoretical Advances and Open Questions on Energy Consumption in Ad-Hoc Wireless Networks. In: Proceedings of the 3rd Workshop on Approximation and Randomization Algorithms in Communication Networks (ARACNE), pp. 23–38 (2002)

Cleve, R., Ekert, A., Macchiavello, C., Mosca, M.: Quantum algorithms revisited. Proc. Royal Soc. London **A454**, 339–354 (1998)

Cleve, R., Luby, M.: A note on self-testing/correcting methods for trigonometric functions. In: International Computer Science Institute Technical Report TR-90-032, July 1990

Clifford, R., Christodoukalis, M., Crawford, T., Meredith, D., Wiggins, G.: A Fast, Randomised, Maximum Subset Matching Algorithm for Document-Level Music Retrieval. In: Proc. International Conference on Music Information Retrieval (ISMIR 2006), University of Victoria, Canada (2006)

Clinton, K.: Transactions costs and covered interest arbitrage: theory and evidence. J. Politcal Econ. **96**(2), 358–370 (1988)

Coan, B.A., Welch, J.L.: Modular construction of a Byzantine agreement protocol with optimal message bit complexity. Inf. Comput. **97**(1), 61–85 (1992)

Cobbs, A.: Fast approximate matching using suffix trees. In: Proceedings of Symposium on Combinatorial Pattern Matching, 1995, pp. 41–54

Codenotti, B., McCune, B., Varadarajan, K.: Market equilibrium via the excess demand function. In: Proceedings STOC'05, pp. 74–83. ACM, Baltimore (2005)

Codenotti, B., Saberi, A., Varadarajan, K., Ye, Y.: Leontief economies encode nonzero sum two-player games. SODA (2006)

Coelho, L.P., Oliveira, A.L.: Dotted suffix trees: a structure for approximate text indexing. In: SPIRE, 2006, pp. 329–336

Coffman, E., Garey, M., Jr., Johnson, D., Lapaugh, A.: Scheduling file transfers. SIAM J. Comput. **14**(3), 744–780 (1985)

Coffman, E.G., Garey, M.R., Johnson, D.S.: Approximation algorithms for bin packing: a survey. In: Hochbaum, D. (ed.) Approximation Algorithms for NP-hard Problems, pp. 46–93. PWS, Boston (1996)

Coffman, E.G., Garey, M.R., Johnson, D.S.: Dynamic bin packing. SIAM J. Comput. **12**(2), 227–258 (1983)

Coffman, E.G., Graham, R.L.: Optimal scheduling for two processors systems. Acta Informatica **1**, 200–213 (1972)

Coffman, E.G., Kleinrock, L.: Feedback Queueing Models for Time-Shared Systems. J. ACM (JACM) **15**(4), 549–576 (1968)

Coffman Jr, E.G., Courcoubetis, C., Garey, M.R., Johnson, D.S., McGeoch, L.A., Shor, P.W., Weber, R.R., Yannakakis, M.: Fundamental discrepancies between average-case analyses under discrete and continuous distributions. In: Proc. of the 23rd Annual ACM Symposium on Theory of Computing, New York, 1991, pp. 230–240. ACM Press, New York (1991)

Coffman Jr., E.G., Gilbert, E.N.: Paths through a maze of rectangles. Networks **22**, 349–367 (1992)

Coffman Jr., E.G., Johnson, D.S., Shor, P.W., Weber, R.R.: Bin packing with discrete item sizes, part II: Tight bounds on first fit. Random Struct. Algorithms **10**, 69–101 (1997)

Coffman Jr., E.G., So, K., Hofri, M., Yao, A.C.: A stochastic model of bin-packing. Inf. Cont. **44**, 105–115 (1980)

Cohen, B.: Incentives build robustness in bittorrent. In: Proceedings of P2P Economics Workshop, 2003

Cohen, E.: Fast algorithms for constructing t-spanners and paths with stretch t. SIAM J. Comput. **28**, 210–236 (1998)

Cohen, H.: A course in computational algebraic number theory. Graduate Texts in Mathematics, vol. 138. Springer (1993)

Cohen, P.R.: Empirical Methods for Artificial Intelligence. MIT Press, Cambridge (1995)

Coja-Oghlan, A., Goerdt, A., Lanka, A., Schädlich, F.: Techniques from combinatorial approximation algorithms yield efficient algorithms for random 2k-SAT. Theor. Comput. Sci. **329**(1–3), 1–45 (2004)

Cole, R. Hariharan, R.: Faster suffix tree construction with missing suffix links. In: Proceedings of the 30th Annual ACM Symposium on Theory of Computing, 2000, pp. 407–415

Cole, R.: On the dynamic finger conjecture for splay trees II: The proof. SIAM J. Comput. **30**(1), 44–85 (2000)

Cole, R., Farach-Colton, M., Hariharan, R., Przytycka, T., Thorup, M.: An $O(n \log n)$ algorithm for the maximum agreement subtree problem for binary trees. SIAM J. Comput. **30**(5), 1385–1404 (2000)

Cole, R., Gottlieb, L.A., Lewenstein, M.: Dictionary matching and indexing with errors and don't cares. In: Proceedings of Symposium on Theory of Computing, 2004, pp. 91–100

Cole, R., Hariharan, R.: A Fast Algorithm for Computing Steiner Edge Connectivity. In: Proc. of the 35th Annual ACM Symposium on Theory of Computing, San Diego 2003, pp. 167–176

Cole, R., Hariharan, R.: An $O(n \log n)$ algorithm for the maximum agreement subtree problem for binary trees. Proc. of the 7th ACM-SIAM SODA, pp. 323–332 (1996)

Cole, R., Hariharan, R.: Approximate string matching: A simpler faster algorithm. SIAM J. Comput. **31**(6), 1761–1782 (2002)

Cole, R., Hariharan, R., Paterson, M., Zwick, U.: Tighter lower bounds on the exact complexity of string matching. SIAM J. Comput. **24**(1), 30–45 (1995)

Cole, R., Klein, P.N., Tarjan, R.E.: Finding minimum spanning forests in logarithmic time and linear work using random sampling. In: Proceedings of the 8th Annual ACM Symposium on Parallel Architectures and Algorithms, 1996, pp. 243–250

Cole, R., Kopelowitz, T., Lewenstein, M.: Suffix trays and suffix trists: Structures for faster text indexing. In: Proc. of International Colloquium on Automata, Languages and Programming (ICALP), 2006, pp. 358–369

Cole, R., Mishra, B., Schmidt, J., Siegel, A.: On the dynamic finger conjecture for splay trees I: Splay sorting $\log n$-block sequences. SIAM J. Comput. **30**(1), 1–43 (2000)

Cole, R., Vishkin, U.: Approximate and Exact Parallel Scheduling with Applications to List, Tree, and Graph Problems. In: Proceedings of the 27th Annual IEEE Symposium on Foundations of Computer Science, 1986, pp. 478–491

Collins, M.: Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In: Conference on Empirical Methods in Natural Language Processing, (2002) Comer, D.E.: The ubiquitous B-tree. ACM Comput. Surv. **11**, 121–137 (1979)

Cominetti, R., Correa, J.R., Moses, N.E.S.: Network games with atomic players. In: Automata, Languages and Programming, 33rd International Colloquium (ICALP), pp. 525–536. Springer, Venice (2006)

Commentz-Walter, B.: A string matching algorithm fast on the average. In: Proceedings of ICALP'79. LNCS, vol. 71, pp. 118–132. Springer, Berlin (1979)

Condon, A., Davy, B., Rastegari, B., Tarrant, F., Zhao, S.: Classifying RNA pseudoknotted structures. Theor. Comput. Sci. **320**, 35–50 (2004)

Cong, J., Ding, Y.: An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs, Proc. IEEE/ACM International Conference on Computer-Aided Design, pp. 48–53. San Jose, USA (1992)

Cong, J., Ding, Y.: Combinational logic synthesis for LUT based field programmable gate arrays. ACM Trans. Design Autom. Electron. Sys. **1**(2): 145–204 (1996)

Cong, J., Ding, Y.: FlowMap: An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs. IEEE Trans. on Comput. Aided Des. of Integr. Circuits and Syst., **13**(1), 1–12 (1994)

Cong, J., Hagen, L., Kahng, A.: Net Partitions Yield Better Module Partitions. In: Proc. 29th ACM/IEEE Design Automation Conf., 1992, pp. 47–52

Cong, J., He, L.: Optimal wiresizing for interconnects with multiple sources. ACM Trans. Des. Autom. Electron. Syst. **1**(4) 568–574 (1996)

Cong, J., Leung, K.-S.: Optimal wiresizing under the distributed Elmore delay model. IEEE Trans. Comput. Des. **14**(3), 321–336 (1995)

Cong, J., Wu, C.: FPGA Synthesis with Retiming and Pipelining for Clock Period Minimization of Sequential Circuits. ACM/IEEE Design Automation Conference (1997)

Conn, A.R., Coulman, P.K., Haring, R.A., Morrill, G.L., Visweshwariah, C., Wu, C.W.: JiffyTune: Circuit Optimization Using Time-Domain Sensitivities. IEEE Trans. Comput. Aided. Des. Integr. Circuits. Syst.**17**(12), 1292–1309 (1998)

Conway, J.H.: An enumeration of knots and links, and some of their algebraic properties. Computational Problems in Abstract Algebra (Proc. Conf., Oxford, 1967), 329–358 (1970)

Conway, J.H., Sloane, N.J.A.: Sphere Packings, Lattices and Groups. Springer, New York (1988)

Cook, S.: The complexity of theorem-proving procedures. In: Proceedings of the 3rd Annual Symposium on Theory of Computing, pp. 151–158. Shaker Heights. May 3–5, 1971.

Cook, S.A., Dwork, C., Reischuk, R.: Upper and lower time bounds for parallel random access machines without simultaneous writes. SIAM J. Comput. **15**(1), 87–97 (1986)

Cook, W., Seymour, P.D.: Tour merging via branch-decomposition. Inf. J. Comput. **15**, 233–248 (2003)

Cook, W.J., Cunningham, W.H., Pulleyblank, W.R., Schrijver, A.: Combinatorial optimization. Wiley, New York (1998)

Coppersmith, D.: Manuscript, private communications (1989)

Coppersmith, D.: Rectangular matrix multiplication revisited. J. Complex. **13**, 42–49 (1997)

Coppersmith, D., Winograd, S.: Matrix Multiplication via Arithmetic Progressions. In: Proceedings of the 19th Annual ACM Conference on Theory of Computing (STOC), 1987, pp. 1–6

Coppersmith, D., Winograd, S.: Matrix multiplication via arithmetic progressions. J. Symb. Comput. **9**(3), 251–280 (1990)

Cormack, G.: Data compression in a data base system. Commun. ACM **28**, 1336–1350 (1985)

Cormen, T.H., Leiserson, C.E., Rivest, R.L.: Introduction to algorithms. McGraw-Hill, New York (1990)

Cormen, T.H., Leiserson, C.E., Rivest, R.L.: Introduction to Algorithms. MIT Press, Cambridge (1989)

Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 2nd edn. MIT Press, Cambridge (2001)

Cormen, T.H., Sundquist, T., Wisniewski, L.F.: Asymptotically tight bounds for performing BMMC permutations on parallel disk systems. SIAM J. Comput. **28**, 105–136 (1999)

Cormode, G., Muthukrishnan, S.: Combinatorial algorithms for compressed sensing. In: Structural Information and Communication Complexity, 13th International Colloquium, SIROCCO (2006), Chester, UK, July 2–5, 2006 pp. 280–294

Cormode, G., Muthukrishnan, S.: Substring compression problems. In: Proc. 16th ACM-SIAM Symposium on Discrete Algorithms (SODA '05), pp. 321–330 (2005)

Cormode, G., Muthukrishnan, S.: The string edit distance matching problem with moves. Proc. ACM-SIAM SODA 667–676 (2002)

Cormode, G., Paterson, M., Sahinalp, S.C., Vishkin, U.: Communication complexity of document exchange. Proc. ACM-SIAM SODA 197–206 (2000)

Cornuéjols, G., Fisher, M.L., Nemhauser, G.L.: Location of bank accounts to optimize float: An analytic study of exact and approximate algorithms. Manag. Sci. **8**, 789–810 (1977)

Cornuejols, G., Nemhauser, G.L., Wolsey, L.A.: The uncapacitated facility location problem. In: Discrete Location Theory, pp. 119–171. Wiley, New York (1990)

Cortes, C., Vapnik, V.: Support-vector network. Mach. Learn. **20**, 273–297 (1995)

Cory, D.G., Fahmy, A.F., Havel, T.F.: Ensemble quantum computing by NMR spectroscopy. Proc. Natl. Acad. Sci. **94**, 1634–1639 (1997)

Cottle, R., Pang, J.S., Stone, R.E.: The linear complementarity problem. Academic Press, Boston (1992)

Coudert, O., Madre, J.C.: New ideas for solving covering problems. In: Proc. Design Automation Conf., 1995, pp. 641–646

Courcoubetis, C., Weber, R.R.: Necessary and sufficient conditions for stability of a bin packing system. J. Appl. Prob. **23**, 989–999 (1986)

Cournier, A., Datta, A.K., Petit, F., Villain, V.: Snap-Stabilizing PIF Algorithm in Arbitrary Networks. In: Proceedings of the 22nd International Conference Distributed Computing Systems, pp. 199–206, Vienna, July 2002

Cover, T.M., Thomas, J.A.: Elements of Information Theory. Wiley Interscience, New York, USA (1990)

Cox, I.J., Rao, S. B., Zhong, Y.: 'Ratio Regions': A Technique for Image Segmentation. In: Proceedings International Conference on Pattern Recognition, IEEE, pp. 557–564, August (1996)

Crammer, K., Dekel, O., Keshet, J., Shalev-Shwartz, S., Singer, Y.: Online passive aggressive algorithms. J. Mach. Learn. Res. **7** (2006)

Crammer, K., Singer, Y.: A new family of online algorithms for category ranking. In: Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (2002)

Cramton, P., Shoham, Y., Steinberg, R.: Combinatorial Auctions. MIT Press (2005)

Cramton, P., Steinberg, R., Shoham, Y. (eds.): Combinatorial Auctions. MIT Press, Cambridge (2005)

Crane, C.A.: Linear lists and priority queues as balanced binary trees. Technical Report STAN-CS-72-259, Computer Science Dept., Stanford University (1972)

Crescenzi, P., Demetrescu, C., Finocchi. I., Petreschi, R.: Reversible Execution and Visualization of Programs with LEONARDO. J. Visual Lang. Comp. **11**, 125–150 (2000). Leonardo is available at: http://www.dis.uniroma1.it/~demetres/Leonardo/. Acessed 15 Jan 2008

Crescenzi, P., Grossi, R., Italiano, G.: Search data structures for skewed strings. In: International Workshop on Experimental and Efficient Algorithms (WEA). Lecture Notes in Computer Science, vol. 2, pp. 81–96. Springer, Berlin (2003)

Cristian, F.: Synchronous atomic broadcast for redundant broadcast channels. Real-Time Syst. **2**, 195–212 (1990)

Cristian, F., Aghili, H., Strong, R., Dolev, D.: Atomic Broadcast: From simple message diffusion to Byzantine agreement. In: Proc. 15th Intl. Symp. on Fault-Tolerant Computing (FTCS-15), Ann Arbor, June 1985 pp. 200–206. IEEE Computer Society Press

Cristian, F., Aghili, H., Strong, R., Dolev, D.: Atomic broadcast: From simple message diffusion to Byzantine agreement. Inform. Comput. **118**, 158–179 (1995)

Cristianini, N., Shawe-Taylor, J.: An Introduction to Support Vector Machines and other kernel-based learning methods. Cambridge University Press, Cambrigde, Book website: www.support-vector.net (2000)

Crochemore, M. : Transducers and repetitions. Theor. Comput. Sci. **45**(1), 63–86 (1986)

Crochemore, M.: An optimal algorithm for computing the repetitions in a word. Inform. Process. Lett. **12**(5), 244–250 (1981)

Crochemore, M.: Recherche linéaire d'un carré dans un mot. Comptes Rendus Acad. Sci. Paris Sér. I Math. **296**, 781–784 (1983)

Crochemore, M., Czumaj, A., Gąsieniec, L., Jarominek, S., Lecroq, T., Plandowski, W., Rytter, W.: Speeding up two string matching algorithms. Algorithmica **12**(4/5), 247–267 (1994)

Crochemore, M., Czumaj, A., Gąsieniec, L., Lecroq, T., Plandowski, W., Rytter, W.: Fast practical multi-pattern matching. Inf. Process. Lett. **71**(3–4), 107–113 (1999)

Crochemore, M., Galil, Z., Gasieniec, L., Hariharan, R., Muthukrishnan, S., Park, K., Ramesh, H., Rytter, W.: Parallel two-dimensional pattern matching. In: Proceeding of 34th Annual IEEE FOCS, 1993, pp. 248–258

Crochemore, M., Hancart, C., Lecroq, T.: Algorithms on strings. Cambridge University Press, Cambridge (2007)

Crochemore, M., Hermelin, D., Landau, G., Rawitz, D., Vialette, S.: Approximating the 2-interval pattern problem, Theoretical Computer Science (special issue for Alberto Apostolico) (2008)

Crochemore, M., Ilie, L.: Analysis of maximal repetitions in strings. J. Comput. Sci. (2007)

Crochemore, M., Landau, G.M., Schieber, B., Ziv-Ukelson, M.: Re-Use Dynamic Programming for Sequence Alignment: An Algorithmic Toolkit. In: Iliopoulos, C.S., Lecroq, T. (eds.) String Algorithmics, pp. 19–59. King's College London Publications, London (2005)

Crochemore, M., Landau, G.M., Ziv-Ukelson, M.: A subquadratic sequence alignment algorithm for unrestricted scoring matrices. SIAM J. Comput. **32**(6), 1654–1673 (2003)

Crochemore, M., Perrin, D.: Two-way string matching. J. ACM **38**(3), 651–675 (1991)

Crochemore, M., Rytter, W.: Jewels of Stringology. World Scientific Publishing Company, Singapore (2002)

Crochemore, M., Rytter, W.: Squares, cubes, and time-space efficient string searching. Algorithmica **13**(5), 405–425 (1995)

Crourant, R., Robbins, H.: What Is Mathematics? Oxford University Press, New York (1941)

Crovella, M.E., Frangioso, R., Harchal-Balter, M.: Connection scheduling in web servers. In: Proceedings of the 2nd USENIX Symposium on Internet Technologies and Systems (USITS-99), 1999 pp. 243–254

Csirik, J., Johnson, D.S.: Bounded space on-line bin packing: Best is better than first. Algorithmica **31**, 115–138 (2001)

Csirik, J., Johnson, D.S., Kenyon, C., Orlin, J.B., Shor, P.W., Weber, R.R.: On the sum-of-squares algorithm for bin packing. J. ACM **53**, 1–65 (2006)

Csirik, J., Johnson, D.S., Kenyon, C., Shor, P.W., Weber, R.R.: A self organizing bin packing heuristic. In: Proc. of the 1999 Workshop on Algorithm Engineering and Experimentation. LNCS, vol. 1619, pp. 246–265. Springer, Berlin (1999)

Csirik, J., Woeginger, G.: On-line packing and covering problems. In: Fiat, A., Woeginger, G. (eds.) Online Algorithms: The State of the Art. LNCS, vol. 1442, pp. 147–177. Springer, Berlin (1998)

Csiszár, I., Körner, J.: Broadcast channels with confidential messages. IEEE Trans. Inf. Theory **24**, 339–348 (1978)

Csürös, M.: Fast recovery of evolutionary trees with thousands of nodes. J. Comput. Biol. **9**(2), 277–297 (2002) Conference version at RECOMB 2001

Csűrös, M.: Maximum-scoring segment sets. IEEE/ACM Trans. Comput. Biol. Bioinform. **1**, 139–150 (2004)

Csürös, M., Kao, M.-Y.: Provably fast and accurate recovery of evolutionary trees through Harmonic Greedy Triplets. SIAM J. Comput. **31**(1), 306–322 (2001) Conference version at SODA (1999)

Csűrös, M., Miklós, I.: A probabilistic model for gene content evolution with duplication, loss, and horizontal transfer. In: Lecture Notes in Bioinformatics, Proceedings of RECOMB2006, vol. 3909, pp. 206–220 (2006)

Culberson, J.C.: http://web.cs.ualberta.ca/~joe/Coloring/index.html

Culberson, J.C., Rudnicki, P.: A fast algorithm for constructing trees from distance matrices. Inf. Process. Lett. **30**, 215–220 (1989)

Culik II, K., Wood, D.: A note on some tree similarity measures. Inf. Process. Lett. **15**(1), 39–42 (1982)

Culler, D.E., Karp, R.M., Patterson, D.A., Sahay, A., Schauser, K.E., Santos, E., Subramonian, R., von Eicken,T.: LogP: Towards a realistic model of parallel computation. In: 4th Symp. Principles and Practice of Parallel Programming, pp. 1–12. ACM SIGPLAN (1993)

Culpepper, J.S., Moffat, A.: Enhanced byte codes with restricted prefix properties. In: Consens, M.P., Navarro, G. (eds.) Proc. Symp. String Processing and Information Retrieval. LNCS Volume 3772, pp. 1–12, Buenos Aires, November 2005

Cypher, R., Meyer auf der Heide, F., Scheideler, C., Vöcking, B.: Universal algorithms for store-and-forward and wormhole routing. In: Proceedings of the 28th ACM Symposium on Theory of Computing, pp. 356–365. Philadelphia, Pennsylvania, USA (1996)

Czumaj, A.: Selfish routing on the Internet. In: Leung, J. (ed.) Handbook of Scheduling: Algorithms, Models, and Performance Analysis. CRC Press, Boca Raton, FL, USA (2004)

Czumaj, A., Ergün, F., Fortnow, L., Magen, A., Newman, I., Rubinfeld, R., Sohler, C.: Approximating the Weight of the Euclidean Minimum Spanning Tree in Sublinear Time. SIAM J. Comput. **35**(1), 91–109 (2005)

Czumaj, A., Krysta, P., Vöcking, B.: Selfish traffic allocation for server farms. In: Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC), pp. 287–296 (2002)

Czumaj, A., Lingas, A.: Approximation schemes for minimum-cost *k*-connectivity problems in geometric graphs. In: Gonzalez, T.F. (eds.) Handbook of Approximation Algorithms and Metaheuristics. CRC Press, Boca Raton (2007)

Czumaj, A., Lingas, A.: Fast approximation schemes for Euclidean multi-connectivity problems. In: Proceedings of the 27th International Colloquium on Automata, Languages and Programming. Lect. Notes Comput. Sci. **1853**, 856–868 (2000)

Czumaj, A., Lingas, A.: On approximability of the minimum-cost *k*-connected spanning subgraph problem. Proc. 10th Annual ACM-SIAM Symposium on Discrete Algorithms, Baltimore, 17–19 January 1999, pp. 281–290

Czumaj, A., Lingas, A., Zhao, H.: Polynomial-time approximation schemes for the Euclidean survivable network design problem. Proc. 29th Annual International Colloquium on Automata, Languages and Programming, Malaga, 8–13 July 2002, pp. 973–984

Czumaj, A., Rytter, W.: Broadcasting algorithms in radio networks with unknown topology. J. Algorithms **60**(2), 115–143 (2006)

Czumaj, A., Vöcking, B.: Tight bounds for worst-case equilibria. ACM Trans. Algorithms **3**(1) (2007)

Czumaj, A., Vöcking, B.: Tight bounds for worst-case equilibria. In: Proc. of the 13th ACM-SIAM Symp. on Discr. Alg. (SODA '02), pp. 413–420. SIAM, San Francisco (2002)

Czumaj, A., Zhao, H.: Fault-tolerant geometric spanners. Discret. Comput. Geom. **32**(2), 207–230 (2004)

Dacre, M., Glazebrook, K., Nino-Mora, J.: The achievable region approach to the optimal control of stochastic systems. J. R. Stat. Soc. Series B **61**(4), 747–791 (1999)

Dahlhaus, E., Johnson, D.S., Papadimitriou, C.H., Seymour, P.D., Yannakakis, M.: The Complexity of Multiterminal Cuts. SIAM J. Comp. **23**, 864–894 (1994). Preliminary version in STOC 1992, An extended abstract was first announced in 1983

Dai, Z., Asada, K.: MOSIZ: A Two-Step Transistor Sizing Algorithm based on Optimal Timing Assignment Method for Multi-Stage Complex Gates. In: Proceedings of the 1989 Custom Integrated Circuits Conference, pp. 17.3.1–17.3.4. May 1989

Daley, R.P., Smith, C.H.: On the Complexity of Inductive Inference. Inform. Control **69**(1–3), 12–40 (1986)

Dalli, D., Wilm, A., Mainz, I., Stegar, G.: STRAL: progressive alignment of non-coding RNA using base pairing probability vectors in quadratic time. Bioinformatics **22**(13), 1593–1599 (2006)

Damron, P., Fedorova, A., Lev, Y., Luchangco, V., Moir, M., Nussbaum, D.: Hybrid transactional memory. In: Proc. 12th Symposium on Architectural Support for Programming Languages and Operating Systems, 2006

Dančík, V., Addona, T., Clauser, K., Vath, J., Pevzner, P.: De novo protein sequencing via tandem mass-spectrometry. J. Comput. Biol. **6**, 327–341 (1999)

Dantsin, E., Goerdt, A., Hirsch, E.A., Kannan, R., Kleinberg, J., Papadimitriou, C., Raghavan, P., Schöning, U.: A deterministic $(2 - 2/(k+1))^n$ algorithm for *k*-SAT based on local search. Theor. Comput. Sci. **289**(1), 69–83 (2002)

Dantsin, E., Hirsch, E.A.: Worst-Case Upper Bounds. In: Biere, A., van Maaren, H., Walsh, T. (eds.) Handbook of Satisfiability. IOS Press (2008) To appear

Dantsin, E., Hirsch, E.A., Wolpert, A.: Clause shortening combined with pruning yields a new upper bound for deterministic SAT algorithms. In: Proceedings of CIAC-2006. Lecture Notes in Computer Science, vol. 3998, pp. 60–68. Springer, Berlin (2006)

Dantsin, E., Wolpert, A.: Max SAT for formulas with constant clause density can be solved faster than in $O(2^n)$ time. In: Proc. of the 9th International Conference on Theory and Applications of Satisfiability Testing. LNCS, vol. 4121, pp. 266–276. Springer, Berlin (2006)

Darga, P.T., Liffiton, M.H., Sakallah, K.A., Markov, I.L.: Exploiting Structure in Symmetry Generation for CNF. In: Proceedings of the 41st Design Automation Conference, 2004, pp. 530–534. Source code at http://vlsicad.eecs.umich.edu/BK/SAUCY/

Darringer, J.A., Brand, D., Gerbi, J.V., Joyner, W.H., Trevillyan, L.H.: LSS: Logic Synthesis through Local Transformations. IBM J. Res. Dev. **25**, 272–280 (1981)

Das, B., Bharghavan, V.: Routing in ad-hoc networks using minimum connected dominating sets. In: Proceedings of IEEE International Conference on on Communications (ICC'97), vol. 1, pp. 376–380. Montreal, 8–12 June 1997

Das, G.: The visibility graph contains a bounded-degree spanner. In: Proceedings of the 9th Canadian Conference on Computational Geometry, Kingston, 11–14 August 1997

Das, G., Joseph, D.: Which Triangulations Approximate the Complete Graph? In: Proc. Int. Symp. Optimal Algorithms. LNCS 401, pp. 168–192. Springer, Berlin (1989)

Das, G., Joseph, D.: Which triangulations approximate the complete graph? In: Proceedings of the International Symposium on Optimal Algorithms. Lecture Notes in Computer Science, vol. 401, pp. 168–192. Springer, Berlin (1989)

Das, G., Narasimhan, G.: A fast algorithm for constructing sparse Euclidean spanners. Int. J. Comput. Geom. Appl. **7**, 297–315 (1997)

Das, G., Narasimhan, G., Salowe, J.: A new way to weigh malnourished Euclidean graphs. In: Proceedings of the 6th ACM-SIAM Symposium on Discrete Algorithms, pp. 215–222. San Francisco, 22–24 January 1995

Dasdan, A., Aykanat, C.: Improved Multiple-Way Circuit Partitioning Algorithms. In: Int. ACM/SIGDA Workshop on Field Programmable Gate Arrays, Feb. 1994

DasGupta, B., He, X., Jiang, T., Li, M., Tromp, J.: On the linear-cost subtree-transfer distance. Algorithmica **25**(2), 176–195 (1999)

DasGupta, B., He, X., Jiang, T., Li, M., Tromp, J., Wang, L., Zhang, L.: Computing Distances between Evolutionary Trees. In: Du, D.Z., Pardalos, P.M. (eds.) Handbook of Combinatorial Optimization. Kluwer Academic Publishers, Norwell, **2**, 35–76 (1998)

DasGupta, B., He, X., Jiang, T., Li, M., Tromp, J., Zhang, L.: On Computing the Nearest Neighbor Interchange Distance. In: Du, D.Z., Pardalos, P.M., Wang, J. (eds.) Proceedings of the DIMACS Workshop on Discrete Problems with Medical Applications, DIMACS Series in Discrete Mathematics and Theoretical Computer Science. Am. Math. Soc. **55**, 125–143 (2000)

DasGupta, B., He, X., Jiang, T., Li, M., Tromp, J., Zhang, L.: On distances between phylogenetic trees, 8th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 427–436 (1997)

DasGupta, B., He, X., Jiang, T., Li, M., Tromp, J., Zhang, L.: On Distances between Phylogenetic Trees. In: Proceedings of the Eighth ACM-SIAM Annual Symposium on Discrete Algorithms (SODA), New Orleans, pp. 427–436. SIAM, Philadelphia (1997)

Daskalakis, C., Goldberg, P.W., Papadimitriou, C.H.: The complexity of computing a Nash equilibrium. In: STOC'06: Proceedings of the 38th ACM Symposium on Theory of Computing, 2006, pp. 71–78

Daskalakis, C., Hill, C., Jaffe, A., Mihaescu, R., Mossel, E., Rao, S.: Maximal accurate forests from distance matrices. In: Proc. Research in Computational Biology (RECOMB), pp. 281–295 (2006)

Daskalakis, C., Mehta, A., Papadimitriou, C.: A note on approximate Nash equilibria. In: Proceedings of the 2nd Workshop on Internet and Network Economics (WINE'06), pp. 297–306. Patras, 15–17 December 2006

Daskalakis, C., Mehta, A., Papadimitriou, C: Progress in approximate Nash equilibrium. In: Proceedings of the 8th ACM Conference on Electronic Commerce (EC07), San Diego, 11–15 June 2007

Daskalakis, C., Mossel, E., Roch, S.: Optimal phylogenetic reconstruction. In: Proc. ACM Symposium on Theory of Computing (STOC), pp. 159–168 (2006)

Daskalakis, C., Papadimitriou, C.H.: Three-player games are hard. ECCC, TR05-139 (2005)

Datta, S., Stojmenovic, I., Wu J.: Internal Node and Shortcut Based Routing with Guaranteed Delivery in Wireless Networks. In: Cluster Computing 5, pp 169–178. Kluwer Academic Publishers, Dordrecht (2002)

Davis, M., Logemann, G., Loveland, D.: A machine program for theorem-proving. Commun. ACM **5**, 394–397 (1962)

Davis, M., Putnam, H.: A computing procedure for quantification theory. J. Assoc. Comput. Mach. **7**(4), 201–215 (1960)

Day, W.H.E.: Optimal Algorithms for Comparing Trees with Labeled Leaves. J. Classif. **2**, 7–28 (1985)

de Berg, M., van Kreveld, M., Overmars, M., Schwarzkopf, O.: Computational Geometry – Algorithms and Applications, 2nd edn. Springer, Heidelberg (2000)

De Bonis, A., Gąsieniec, L., Vaccaro, U.: Optimal Two-Stage Algorithms for Group Testing Problems. SIAM J. Comput. **34**(5), 1253–1270 (2005)

de Klerk, E., Pasechnik, D., Warners, J.: On approximate graph colouring and MAX-k-CUT algorithms based on the θ function. J. Combin. Optim. **8**(3), 267–294 (2004)

De Marco, G., Gargano, L., Kranakis, E., Krizanc, D., Pelc, A., Vaccaro, U.: Asynchronous Deterministic Rendezvous in Graphs. Theoret. Comput. Sci. **355**, 315–326 (2006)

De Marco, G., Pelc, A.: Faster broadcasting in unknown radio networks. Inf. Process. Lett. **79**(2), 53–56 (2001)

De Micheli, G.: Synthesis and Optimization of Digital Circuits, 1st edn., pp. 504–533. McGraw-Hill, New York (1994)

de Moura, E.S., Navarro, G., Ziviani, N., Baeza-Yates, R.: Fast and flexible word searching on compressed text. ACM Trans. Inf. Syst. **18**(2), 113–139 (2000)

De Roberts, E., Oliver, G., Wright, C.: Homeobox genes and the vertibrate body plan, pp. 46–52. Scientific American (1990)

Défago, X., Schiper, A., Urbán, P.: Total order broadcast and multicast algorithms: Taxonomy and survey. ACM Comput. Surv. **36**, 372–421 (2004)

Degermark, M., Brodnik, A., Carlsson, S., Pink, S.: Small forwarding tables for fast routing lookups. In: Proc. ACM SIGCOMM, 1997, pp. 3–14

Dehne, F., Fellows, M., Langston, M., Rosamond, F., Stevens, K.: An $O(2^{O(k)}n^3)$ FPT algorithm for the undirected feedback vertex set problem. Proceedings COCOON 2005. Lecture Notes in Computer Science, vol. 3595, pp. 859–869. Springer, Berlin (2005)

Deĭneko, V.G., Hoffmann, M., Okamoto, Y., Woeginger, G.J.: The traveling salesman problem with few inner points. Oper. Res. Lett. **31**, 106–110 (2006)

Dekel, E., Nassimi, D., Sahni, S.: Parallel matrix and graph algorithms. SIAM J. Comput. **10**, 657–675 (1981)

Dekel, O., Shalev-Shwartz, S., Singer, Y.: The Forgetron: A kernel-based perceptron on a fixed budget. In: Advances in Neural Information Processing Systems 18 (2005)

Delgrange, O., Rivals, E.: STAR – an algorithm to Search for Tandem Approximate Repeats. Bioinform. **20**, 2812–2820 (2004)

Delling, D., Holzer, M., Müller, K., Schulz, F., Wagner, D.: High-Performance Multi-Level Graphs. In: 9th DIMACS Challenge on Shortest Paths, Nov 2006. Rutgers University, USA (2006)

Delling, D., Holzer, M., Muller, K., Schulz, F., Wagner, D.: High-performance multi-level graphs. In: 9th DIMACS Implementation Challenge Workshop: Shortest Paths, DIMACS Center, Piscataway, NJ, 13–14 Nov 2006

Delling, D., Sanders, P., Schultes, D., Wagner, D.: Highway Hierarchies Star. In: 9th DIMACS Challenge on Shortest Paths, Nov 2006 Rutgers University, USA (2006)

Delling, D., Sanders, P., Schultes, D., Wagner, D.: Highway hierarchies star. In: 9th DIMACS Implementation Challenge Workshop: Shortest Paths, DIMACS Center, Piscataway, NJ, 13–14 Nov 2006

Delorme, C., Poljak, S.: Laplacian eigenvalues and the maximum cut problem. Math. Program. **62**, 557–574 (1993)

Delorme, C., Poljak, S.: The performance of an eigenvalue bound in some classes of graphs. Discret. Math. **111**, 145–156 (1993). Also appeared in: Proceedings of the Conference on Combinatorics, Marseille, 1990

Delporte-Gallet, C., Fauconnier, H., Guerraoui, R.: Failure detection lower bounds on registers and consensus. In: Proceedings of the 16th International Symposium on Distributed Computing, LNCS 2508 (2002)

Delporte-Gallet, C., Fauconnier, H., Guerraoui, R.: Implementing atomic objects in a message passing system. Technical report, EPFL Lausanne (2005)

Delporte-Gallet, C., Fauconnier, H., Guerraoui, R., Hadzilacos, V., Kouznetsov, P., Toueg, S.: The weakest failure detectors to solve certain fundamental problems in distributed computing. In: Proc. 23rd ACM Symposium on Principles of Distributed Computing, pp. 338–346. St. John's, Newfoundland, 25–28 July 2004

Delpratt, O., Rahman, N., Raman, R.: Compressed prefix sums. In: Proc. SOFSEM 2007. LNCS, vol. 4362, pp. 235–247 (2007)

Delpratt, O., Rahman, N., Raman, R.: Engineering the LOUDS succinct tree representation. In: Proc. WEA 2006. LNCS, vol. 4007, pp. 134–145. Springer, Berlin (2006)

Demaine, E., Fekete, S., Gal, S.: Online searching with turn cost. Theor. Comput. Sci. **361**, 342–355 (2006)

Demaine, E.D.: Cache-oblivious algorithms and data structures. In: Proc. EFF summer school on massive data sets, LNCS. Springer, Berlin. To appear. Online version at http://theory.csail.mit.edu/edemaine/papers/BRICS2002/

Demaine, E.D., Fomin, F.V., Hajiaghayi, M., Thilikos, D.M.: Bidimensional parameters and local treewidth. SIAM J. Discret. Math. **18**(3), 501–511 (2004)

Demaine, E.D., Fomin, F.V., Hajiaghayi, M., Thilikos, D.M.: Fixed-parameter algorithms for (k,r)-center in planar graphs and map graphs. ACM Trans. Algorithms **1**(1), 33–47 (2005)

Demaine, E.D., Fomin, F.V., Hajiaghayi, M., Thilikos, D.M.: Subexponential parametrized algorithms on graphs of bounded genus and H-minor-free graphs. J. ACM **52**(6), 866–893 (2005)

Demaine, E.D., Hajiaghayi, M.: Bidimensionality: new connections between FPT algorithms and PTASs. In: Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2005), Vancouver, January 2005, pp. 590–601

Demaine, E.D., Hajiaghayi, M.: Diameter and treewidth in minor-closed graph families, revisited. Algorithmica **40**(3), 211–215 (2004)

Demaine, E.D., Hajiaghayi, M.: Equivalence of local treewidth and linear local treewidth and its algorithmic applications. In: Proceedings of the 15th ACM-SIAM Symposium on Discrete Algorithms (SODA'04), January 2004, pp. 833–842 (2004)

Demaine, E.D., Hajiaghayi, M.: Graphs excluding a fixed minor have grids as large as treewidth, with combinatorial and algorithmic applications through bidimensionality. In: Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2005), pp. 682–689. Vancouver, January (2005)

Demaine, E.D., Hajiaghayi, M.: The bidimensionality theory and its algorithmic applications. Comput. J. To appear

Demaine, E.D., Hajiaghayi, M., Kawarabayashi, K.-I.: Algorithmic graph minor theory: Decomposition, approximation, and coloring. In: Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science, Pittsburgh, October 2005, pp. 637–646

Demaine, E.D., Hajiaghayi, M., Kawarabayashi, K.: Algorithmic graph minor theory: Improved grid minor bounds and Wagner's contraction. In: Proceedings of the 17th Annual International Symposium on Algorithms and Computation, Calcutta, India, December 2006. Lecture Notes in Computer Science, vol. 4288, pp. 3–15 (2006)

Demaine, E.D., Hajiaghayi, M., Mohar, B.: Approximation algorithms via contraction decomposition. In: Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms, New Orleans, 7–9 January 2007, pp. 278–287

Demaine, E.D., Hajiaghayi, M., Nishimura, N., Ragde, P., Thilikos, D.M.: Approximation algorithms for classes of graphs excluding single-crossing graphs as minors. J. Comput. Syst. Sci. **69**(2), 166–195 (2004)

Demaine, E.D., Hajiaghayi, M., Thilikos, D.M.: Exponential speedup of fixed-parameter algorithms for classes of graphs excluding single-crossing graphs as minors. Algorithmica **41**(4), 245–267 (2005)

Demaine, E.D., Hajiaghayi, M., Thilikos, D.M.: The bidimensional theory of bounded-genus graphs. SIAM J. Discret. Math. **20**(2), 357–371 (2006)

Demaine, E.D., Harmon, D., Iacono, J., Patrascu, M.: Dynamic optimality —almost. SIAM J. Comput. **37**(1), 240–251 (2007)

Demange, G.: Spatial Models of Collective Choice. In: Thisse, J.F., Zoller, H.G. (eds.) Locational Analysis of Public Facilities, North-Holland Publishing Company, North Holland, Amsterdam (1983)

Demange, G., Gale, D., Sotomayor, M.: Multi-item auctions. J. Polit. Econ. **94**(4), 863–72 (1986)

Dementiev, R., Mehnert, J., Kärkkäinen, J., Sanders, P.: Better external memory suffix array construction. ACM J. Exp. Algorithmics (2008) in press

Demestrescu, C., Italiano, G.F.: Trade-offs for fully dynamic transitive closure on DAG's: breaking through the $O(n^2)$ barrier, (presented in FOCS 2000). J. ACM **52**(2), 147–156 (2005)

Demetrescu, C.: Fully Dynamic Algorithms for Path Problems on Directed Graphs. Ph. D. thesis, Department of Computer and Systems Science, University of Rome "La Sapienza", Rome (2001)

Demetrescu, C., Finocchi, I., Italiano, G.: Dynamic Graphs. In: Mehta, D., Sahni, S. (eds.) Handbook on Data Structures and Applications (CRC Press Series, in Computer and Information Science), chap. 36. CRC Press, Boca Raton (2005)

Demetrescu, C., Finocchi, I., Italiano, G.F., Näher, S.: Visualization in algorithm engineering: tools and techniques. In: Experimental Algorithm Design to Robust and Effizient Software. Lecture Notes in Computer Science, vol. 2547. Springer, Berlin, pp. 24–50 (2002)

Demetrescu, C., Finocchi, I., Liotta, G.: Visualizing Algorithms over the Web with the Publication-driven Approach. In: Proc. of the 4th Workshop on Algorithm Engineering (WAE'00), Saarbrücken, Germany, 5–8 September (2000)

Demetrescu, C., Goldberg, A.V., Johnson, D.: 9th DIMACS Implementation challenge – shortest paths. http://www.dis.uniroma1.it/~challenge9/ (2006)

Demetrescu, C., Italiano, G.: Fully dynamic transitive closure: Breaking through the $O(n^2)$ barrier. In: Proc. of the 41st IEEE Annual Symposium on Foundations of Computer Science (FOCS'00), Redondo Beach (2000), pp. 381–389

Demetrescu, C., Italiano, G.: Trade-offs for fully dynamic reachability on dags: Breaking through the $O(n^2)$ barrier. J. ACM **52**, 147–156 (2005)

Demetrescu, C., Italiano, G.F.: A new approach to dynamic all pairs shortest paths. J. Assoc. Comp. Mach. **51**(6), 968–992 (2004)

Demetrescu, C., Italiano, G.F.: Experimental analysis of dynamic all pairs shortest path algorithms. ACM Trans. Algorithms **2**(4), 578–601 (2006)

Demetrescu, C., Italiano, G.F.: Fully Dynamic All Pairs Shortest Paths with Real Edge Weights. J. Comp. Syst. Sci. **72**(5), 813–837 (2006)

Deng, X.: Combinatorial Optimization and Coalition Games. In: Du, D., Pardalos, P.M. (eds.) Handbook of combinatorial optimization, vol 2, pp 77–103, Kluwer, Boston (1998)

Deng, X., Fang, Q., Sun, X.: Finding Nucleolus of Flow Games. Proceedings of the 17th annual ACM-SIAM symposium on Discrete algorithm (SODA 2006). Lect. Notes in Comput. Sci. **3111**, 124–131 (2006)

Deng, X., Huang, L.-S., Li, M.: On Walrasian Price of CPU time. In: Proceedings of COCOON'05, Knming, 16–19 August 2005, pp. 586–595. Algorithmica **48**(2), 159–172 (2007)

Deng, X., Ibaraki, T., Nagamochi, H.: Algorithmic Aspects of the Core of Combinatorial Optimization Games. Math. Oper. Res. **24**, 751–766 (1999)

Deng, X., Kameda, T., Papadimitriou, C.H.: How to learn an unknown environment. J. ACM **45**, 215–245 (1998)

Deng, X., Li, G., Li, Z., Ma, B., Wang, L.: Genetic Design of Drugs Without Side-Effects. SIAM. J. Comput. **32**(4), 1073–1090 (2003)

Deng, X., Li, Z.F., Wang, S.: Computational complexity of arbitrage in frictional security market. Int. J. Found. Comput. Sci. **13**(5), 681–684 (2002)

Deng, X., Papadimitriou, C.: On the complexity of cooperative game solution concepts. Math. Oper. Res. **19**(2), 257–266 (1994)

Deng, X., Papadimitriou, C., Safra, S.: On the complexity of price equilibria. J. Comput. System Sci. **67**(2), 311–324 (2003)

Deng, X., Papadimitriou, C.H.: Exploring an unknown graph. J. Graph Theory **32**, 265–297 (1999)

Denne, E., Sullivan, J.M.: The Distortion of a Knotted Curve. http://www.arxiv.org/abs/math.GT/0409438 (2004)

Deo, N., Prabhu, G.M., Krishnamoorthy, M.S.: Algorithms for generating fundamental cycles in a graph. ACM Trans. Math. Softw. **8**, 26–42 (1982)

Department of Computer Science, Duke University. TPIE: a transparent parallel I/O environment. http://www.cs.duke.edu/TPIE/. Accessed 2002

Derryberry, J., Sleator, D.D., Wang, C.C.: A lower bound framework for binary search trees with rotations. Technical Report CMU-CS-05-187, Carnegie Mellon University (2005)

Deshmukh, K., Goldberg, A.V., Hartline, J.D., Karlin, A.R.: Truthful and competitive double auctions. In: Möhring, R.H., Raman, R. (eds.) Algorithms–ESA 2002, 10th Annual European Symposium, Rome, Italy, 17–21 Sept 2002. Lecture Notes in Computer Science, vol. 2461, pp. 361–373. Springer, Berlin (2002)

Desper, R., Gascuel, O.: Fast and Accurate Phylogeny Reconstruction Algorithms Based on the Minimum – Evolution Principle. J. Comput. Biol. **9**, 687–706 (2002)

Dessmark, A., Pelc, A.: Broadcasting in geometric radio networks. J. Discret. Algorithms **5**, 187–201 (2007)

Dessmark, A., Pelc, A.: Tradeoffs between knowledge and time of communication in geometric radio networks. Proc. 13th Ann. ACM Symposium on Parallel Algorithms and Architectures (SPAA), pp. 59–66, Crete Greece, July 3–6, 2001

Dette, H., Henze, N.: The limit distribution of the largest nearest-neighbour link in the unit *d*-cube. J. Appl. Probab. **26**, 67–80 (1989)

Deutsch, D.: Quantum theory, the Church-Turing principle and the universal quantum computer. Proc. Royal Soc. London A **400**, 97–117 (1985)

Deutsch, D., Jozsa, R.: Rapid solutions of problems by quantum computation. Proc. Royal Soc. London A **439**, 553–558 (1992)

Devadas, S., Ghosh, A., Keutzer, K.: Logic Synthesis. McGraw Hill, New York (1994). pp. 185–200

Devanur, N.R., Khot, S.A., Saket, R., Vishnoi, N.K.: Integrality gaps for sparsest cut and minimum linear arrangement problems. In: STOC '06: Proceedings of the thirty-eighth annual ACM symposium on Theory of computing, pp. 537–546. ACM Press, New York (2006)

Devanur, N.R., Papadimitriou, C.H., Saberi, A., Vazirani, V.V.: Market equilibria via a primal-dual-type algorithm. In: Proceedings of FOCS'02, pp. 389–395. IEEE Computer Society, Vancouver (2002)

Devetak, I., Harrow, A., Winter, A.: A resource framework for quantum Shannon theory. Tech. Report CSTR-05-008, CS Department, University of Bristol, December (2005)

Devetak, I., Winter, A.: Distillation of secret key and entanglement from quantum states. Proc. R. Soc. Lond. A **461**, 207–235 (2005)

DeVos, M., Ding, G., Oporowski, B., Sanders, D.P., Reed, B., Seymour, P., Vertigan, D.: Excluding any graph as a minor allows a low tree-width 2-coloring. J. Comb. Theory Ser. B **91**(1), 25–41 (2004)

Devroye, L.: Non-uniform Random Variate Generation. Springer, New York (1986)

Devroye, L., Gyorfi, L., Lugosi, G.: A Probabilistic Theory of Pattern Recognition. Springer, New York, USA (1996)

DeWitt, D.J., Kabra, N., Luo, J., Patel, J.M., Yu, J.-B.: Client-server paradise. In: Proc. International Conference on Very Large Databases, 1994, pp. 558–569

Dey, T.K.: Curve and surface reconstruction. In: Goodman, J.E., O'Rourke, J. (eds.) Handbook of Discrete and Computational Geometry, 2nd edn. CRC, Boca Raton (2004)

Dey, T.K.: Curve and Surface Reconstruction: Algorithms with Mathematical Analysis. Cambridge University Press, New York (2006)

Dhagat, A., Hellerstein, L.: PAC learning with irrelevant attributes. In: Proceedings of the 35th Annual Symposium on Foundations of Computer Science, Santa Fe, pp 64–74. IEEE Computer Society, Los Alamitos (1994)

Diao, Y., Fischer, P., Franklin, M., To, R.: YFilter: Efficient and scalable filtering of XML documents. In: *Proceedings of the 18th International Conference on Data Engineering*, San Jose, California, pp. 341–342. IEEE Computer Society, New Jersey (2002)

Diaz-Gutierrez, P., Bhushan, A., Gopi, M., Pajarola, R.: Single-strips for fast interactive rendering. J. Vis. Comput. **22**(6), 372–386 (2006)

Díaz, J., Serna, M., Spirakis, P.G., Torán, J.: Paradigms for fast parallel approximation. In: Cambridge International Series on Parallel Computation, vol 8, Cambridge University Press, Cambridge (1997)

Diekmann, Y., Sagot, M.F., Tannier, E.: Evolution under Reversals: Parsimony and Conversation of Common Intervals. IEEE/ACM Transactions in Computational Biology and Bioinformatics, **4**, 301–309, 1075 (2007)

Dietzfelbinger, M., Weidling, C.: Balanced allocation and dictionaries with tightly packed constant size bins. In: ICALP. Lecture Notes in Computer Science, vol. 3580, pp. 166–178. Springer, Berlin (2005)

Diffie, W., Hellman, M.: New directions in cryptography. IEEE Trans. Inf. Theor. **22**, 644–654 (1976)

Diggavi, S.N., Grossglauser, M., Tse, D.N.C.: Even one-dimensional mobility increases the capacity of wireless networks. IEEE Trans. Inf. Theory **51**(11), 3947–3954 (2005)

Dijkstra, E.W.: A note on two problems in connexion with graphs. Numer. Math. **1** 269–271 (1959)

Dijkstra, E.W.: Co-operating sequential processes. In: Genuys, F. (ed.) Programming Languages, pp. 43–112. Academic Press, New York (1968). Reprinted from: Technical Report EWD-123, Technological University, Eindhoven (1965)

Dijkstra, E.W.: Self Stabilizing Systems in Spite of Distributed Control. Commun. ACM **17**(11), 643–644 (1974). See also EWD391 (1973) In: Selected Writings on Computing: A Personal Perspective, pp. 41–46. Springer, New York (1982)

Dijkstra, E.W.: Solution of a problem in concurrent programming control. Commun. ACM **8**(9), 569 (1965)

Diks, K., Fraigniaud, P., Kranakis, E., Pelc, A.: Tree exploration with little memory. J. Algorithms **51**, 38–63 (2004)

Diks, K., Kranakis, E., Krizanc, D., Pelc, A.: The impact of knowledge on broadcasting time in linear radio networks. Theor. Comput. Sci. **287**, 449–471 (2002)

Dilley, J., Arlitt, M., Perret, S.: Enhancement and validation of Squid's cache replacement policy. Hewlett-Packard Laboratories Technical Report HPL-1999–69 (1999)

DIMACS Implementation Challenges. Each DIMACS Implementation Challenge is a year-long cooperative research event in which researchers cooperate to find the most efficient algorithms and strategies for selected algorithmic problems. The DIMACS Challenges since 1991 have targeted a variety of optimization problems on graphs; advanced data structures; and scientific application areas involving computational biology and parallel computation. The DIMACS Challenge proceedings are published by AMS as part of theDIMACS Series in Discrete

Mathematics and Theoretical Computer Science. Visit dimacs.rutgers.edu/Challenges for more information

Dimitriou, T., Nikoletseas, S.E., Spirakis, P.G.: Analysis of the information propagation time among mobile hosts. In: Nikolaidis, I., Barbeau, M., Kranakis, E. (eds.) 3rd International Conference on Ad-Hoc, Mobile, and Wireless Networks (ADHOC-NOW 2004), pp 122–134. Lecture Notes in Computer Science (LNCS), vol. 3158. Springer, Berlin (2004)

Dimitrov, D., Borm, P., Hendrickx, R., Sung, S. Ch.: Simple priorities and core stability in hedonic games. Soc. Choice. Welf. **26**(2), 421–433 (2006)

Ding, Y., Chan, C.Y., Lawrence, C.E.: RNA secondary structure prediction by centroids in a Boltzmann weighted ensemble. RNA **11**, 1157–1166 (2005)

Ding, Z., Filkov, V., Gusfield, D.: A linear-time algorithm for the perfect phylogeny haplotyping problem. In: Proceedings of the Annual International Conference on Computational Molecular Biology (RECOMB), New York, 2005. ACM Press, New York (2005)

Dinitz, E.A.: Maintaining the 4-edge-connected components of a graph on-line. In: Proc. 2nd Israel Symp. Theory of Computing and Systems, 1993, pp. 88–99

Dinitz, E.A., Karzanov A.V., Lomonosov M.V.: On the structure of the system of minimal edge cuts in a graph. In: Fridman, A.A. (ed) Studies in Discrete Optimization, pp. 290–306. Nauka, Moscow (1990). In Russian

Dinur, I., Kindler, G., Raz, R., Safra, S.: Approximating CVP to within almost-polynomial factors is NP-hard. Combinatorica **23**(2), 205–243 (2003). Preliminary version in FOCS 1998

Dinur, I., Mossel, E., Regev, O.: Conditional hardness for approximate coloring. In: Proceedings of the 38th annual ACM Symposium on Theory of Computing (2006) pp. 344–353.

Dirks, R.M., Pierce, N.A.: A partition function algorithm for nucleic acid secondary structure including pseudoknots. J. Comput. Chem. **24**, 1664–1677 (2003)

Dixon, B., Rauch, M., Tarjan, R.E.: Verification and sensitivity analysis of minimum spanning trees in linear time. SIAM J. Comput. **21**(6), 1184–1192 (1992)

Dobkin, D.P., Friedman, S.J., Supowit, K.J.: Delaunay Graphs Are Almost as Good as Complete Graphs. Discret. Comput. Geom. **5**, 399–407 (1990)

Dobzinski, S., Nisan, N., Schapira, M.: Truthful randomized mechanisms for combinatorial auctions. In: Proc. of the 38th ACM Symposium on Theory of Computing (STOC'06), 2006

Dolev, D., Dwork, C., Stockmeyer, L.: On the minimal synchrony needed for distributed consensus. J. ACM **34**(1), 77–97 (1987)

Dolev, D., Reischuk, R.: Bounds on Information Exchange for Byzantine Agreement. J. ACM **32**(1), 191–204 (1985)

Dolev, D., Reischuk, R., Strong, H.R.: Early Stopping in Byzantine Agreement. J. ACM **37**(4), 720–741 (1990)

Dolev, D., Shavit, N.: Bounded concurrent time-stamp systems are constructible. SIAM J. Comput. **26**(2), 418–455 (1997)

Dolev, D., Strong, H.R.: Authenticated Algorithms for Byzantine Agreement. SIAM J. Comput. **12**(4), 656–666 (1983)

Dolev, S.: Self-Stabilization. MIT Press, Cambrigde (2000)

Dolev, S., Gilbert, S., Lynch, N.A., Shvartsman, A.A., Welch, J.L.: GeoQuorums: Implementing atomic memory in mobile ad hoc networks. Distrib. Comput. **18**(2), 125–155 (2005)

Dolev, S., Gouda, M.G., Schneider, M.: Memory Requirements for Silent Stabilization. In: Proceedings of the 15th Annual ACM Symposium on Principles of Distributed Computing, pp. 27–34, Philadelphia, May 1996

Dolev, S., Yagel, R.: Toward Self-Stabilizing Operating Systems. In: 2nd International Workshop on Self-Adaptive and Autonomic Computing Systems, pp. 684–688, Zaragoza, August 2004

Dopazo, J., Rodríguez, A., Sáiz, J.C., Sobrino, F.: Design of primers for PCR amplification of highly variable genomes. CABIOS **9**, 123–125 (1993)

Dor, D., Halperin, S., Zwick, U.: All Pairs Almost Shortest Paths. SIAM J. Comput. **29**, 1740–1759 (2000)

Dorn, F.: Dynamic Programming and Fast Matrix Multiplication. In: Proceedings of 14th Annual European Symposium on Algorithms. LNCS, vol. 4168, pp. 280–291. Springer, Berlin (2006)

Dorn, F., Fomin, F.V., Thilikos, D.M.: Fast subexponential algorithm for non-local problems on graphs of bounded genus. In: Proceedings of the 10th Scandinavian Workshop on Algorithm Theory (SWAT 2006). Lecture Notes in Computer Science. Springer, Berlin (2005)

Dorn, F., Fomin, F.V., Thilikos, D.M.: Subexponential algorithms for non-local problems on *H*-minor-free graphs. In: Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms (SODA 2008). pp. 631–640. Society for Industrial and Applied Mathematics, Philadelphia (2006)

Dorn, F., Penninkx, E., Bodlaender, H., Fomin, F.V.: Efficient exact algorithms on planar graphs: Exploiting sphere cut branch decompositions. In: Proceedings of the 13th Annual European Symposium on Algorithms (ESA 2005). Lecture Notes in Computer Science, vol. 3669, pp. 95–106. Springer, Berlin (2005)

Douligeris, C., Mazumdar, R.: Multilevel flow control of Queues. In: Johns Hopkins Conference on Information Sciences, Baltimore, 22–24 Mar 1989 (2006)

Dowell, R., Eddy, S.R.: Evaluation of several lightweight stochastic context-free grammars for RNA secondary structure prediction. BMC Bioinformatics **5**, 71 (2004)

Downey, R.G., Fellows, M.R.: Fixed-parameter tractability and completeness. Congres. Numerant. **87**, 161–187 (1992)

Downey, R.G., Fellows, M.R.: Parameterized complexity. In: Monographs in Computer Science. Springer, New York (1999)

Dress, A., Steel, M.: Convex tree realizations of partitions. Appl. Math. Lett. **5**, 3–6 (1992)

Driscoll, J.R., Sarnak, N., Sleator, D.D., Tarjan, R.E.: Making data structures persistent. J. Comput. Syst. Sci. **38**(1), 86–124 (1989). See also STOC'86

Driscoll, K., Hall, B., Sivencrona, H., Zumsteg, P.: Byzantine Fault Tolerance, from Theory to Reality. In: Proc. 22nd International Conference on Computer Safety, Reliability, and Security (SAFECOMP), pp. 235–248, UK, September 2003

Drucker, H., Burges, C.J.C., Kaufman, L., Smola, A., Vapnik, V.: Support Vector Regression Machines. Adv. Neural. Inf. Process. Syst. (NIPS) **9**, 155–161 MIT Press (1997)

Drysdale, R.L., McElfresh, S., Snoeyink, J.S.: On exclusion regions for optimal triangulations. Discrete Appl. Math. **109**, 49–65 (2001)

Du, D.-Z., Hsu, D.F., Xu, K.-J.: Bounds on guillotine ratio. Congressus Numerantium **58**, 313–318 (1987)

Du, D.-Z., Pan, L.-Q., Shing, M.-T.: Minimum edge length guillotine rectangular partition. Technical Report 0241886, Math. Sci. Res. Inst., Univ. California, Berkeley (1986)

Du, D.Z., Graham, R.L., Pardalos, P.M., Wan, P.J., Wu, W., Zhao, W.: Analysis of greedy approximations with nonsubmodular potential functions. In: Proceedings of 19th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 167–175. ACM, New York (2008)

Du, D.Z., Hwang, F.K.: The Steiner ratio conjecture of Gilbert-Pollak is true. Proc. Natl. Acad. Sci. USA **87**, 9464–9466 (1990)

Du, D.Z., Hwang, F.K., Shing, M.T., Witbold, T.: Optimal routing trees. IEEE Trans. Circuits **35**, 1335–1337 (1988)

Du, D.Z., Zhang, Y., Feng, Q.: On better heuristic for euclidean Steiner minimum trees. In: Proceedings 32nd FOCS, IEEE Computer Society Press, California (1991)

Dubhashi, D., Mei, A., Panconesi, A., Radhakrishnan, J., Srinivasan, A.: Fast Distributed Algorithms for (Weakly) Connected Dominating Sets and Linear-Size Skeletons. In: SODA, 2003, pp. 717–724

Dubois, O.: Upper bounds on the satisfiability threshold. Theor. Comput. Sci. **265**, 187–197 (2001)

Dubois, O., Boufkhad, Y., Mandler, J.: Typical random 3-sat formulae and the satisfiability threshold. In: 11th ACM-SIAM symposium on Discrete algorithms, pp. 126–127. Society for Industrial and Applied Mathematics, San Francisco (2000)

Duda, R.O., Hart, P.E., Stork, D.G.: Pattern Classification. Wiley-Interscience Publication (2000)

Dudek, G., Romanik, K., Whitesides, S.: Localizing a robot with minimum travel. SIAM J. Comput. **27**(2), 583–604 (1998)

Duffin, R.J.: Topology of Series-Parallel Networks. J. Math. Anal. Appl. **10**, 303–318 (1965)

Dujmović, V., Fellows, M.R., Hallett, M., Kitching, M., Liotta, G., McCartin, C., Nishimura, N., Ragde, P., Rosamond, F.A., Suderman, M., Whitesides, S., Wood, D.R.: A fixed-parameter approach to 2-layer planarization. Algorithmica **45**, 159–182 (2006)

Dujmović, V., Fernau, H., Kaufmann, M.: Fixed parameter algorithms for one-sided crossing minimization revisited. In: Liotta G. (ed.) Graph Drawing, 11th International Symposium GD 2003. LNCS, vol. 2912, pp. 332–344. Springer, Berlin (2004). A journal version has been accepted to J. Discret. Algorithms, see doi: 10.1016/j.jda.2006.12.008

Dujmović, V., Whitesides, S.: An efficient fixed parameter tractable algorithm for 1-sided crossing minimization. Algorithmica **40**, 15–32 (2004)

Dumais, S., Platt, J., Heckerman, D., Sahami, M.: Inductive learning algorithms and representations for text categorization. In: 7th International Conference on Information and Knowledge Management (1998)

Dumitrescu, A., Ebbers-Baumann, A., Grüne, A., Klein, R., Rote, G.: On the Geometric Dilation of Closed Curves, Graphs, and Point Sets. Comput. Geom. Theory Appl. **36**(1), 16–38 (2006)

Dunagan, J., Vempala, S.: On Euclidean embeddings and bandwidth minimization. In: Randomization, approximation, and combinatorial optimization, pp. 229–240. Springer (2001)

Durbin, R., Eddy, S., Krogh, A., Mitchison, G.: Biological sequence analysis. Cambridge University Press, Cambridge, UK (1998)

Dutta, P., Guerraoui, R., Levy, R.R., Chakraborty, A.: How fast can a distributed atomic read be? In: Proc. 23rd ACM Symposium on Principles of Distributed Computing, pp. 236–245. St. John's, Newfoundland, 25–28 July 2004

Dutta, R., Savage, C.: A Note on the Complexity of Converter Placement Supporting Broadcast in WDM Optical Networks. In: Proceedings of the International Conference on Telecommunication Systems-Modeling and Analysis, Dallas, November 2005 ISBN: 0-9716253-3-6 pp. 23–31. American Telecommunication Systems Management Association, Nashville

Dwork, C., Lynch, N.A., Stockmeyer, L.: Consensus in the presence of partial synchrony. J. ACM **35**(2), 288–323 (1988)

Dwork, C., Moses, Y.: Knowledge and Common Knowledge in a Byzantine Environment: Crash Failures. Inf. Comput. **88**(2), 156–186 (1990)

Dyachkov, A.G., Rykov, V.V.: Bounds on the length of disjunctive codes. Problemy Peredachi Informatsii **18**(3), 7–13 (1982)

Eades, P., Wormald, N.C.: Edge crossings in drawings of bipartite graphs. Algorithmica **11**, 379–403 (1994)

Eaves, B.C.: Finite solution for pure trade markets with Cobb-Douglas utilities, Math. Program. Study **23**, 226–239 (1985)

Ebbers-Baumann, A., Gruene, A., Karpinski, M., Klein, R., Knauer, C., Lingas, A.: Embedding Point Sets into Plane Graphs of Small Dilation. Int. J. Comput. Geom. Appl. **17**(3), 201–230 (2007)

Ebbers-Baumann, A., Grüne, A., Klein, R.: On the Geometric Dilation of Finite Point Sets. Algorithmica **44**(2), 137–149 (2006)

Ebbers-Baumann, A., Klein, R., Knauer, C., Rote, G.: The Geometric Dilation of Three Points. Manuscript (2006)

Economides, A., Silvester, J.: Priority load sharing: an approach using stackelberg games. In: 28th Annual Allerton Conference on Communications, Control and Computing (1990)

Edelman, B., Ostrovsky, M., Schwartz, M.: Internet advertising and the generalized second price auction. NBER Working Paper, 11765, November 2005

Edelman, B., Ostrovsky, M., Schwarz, M.: Internet advertising and the generalized second price auction: selling billions of dollars worth of dollars worth of keywords. In: 2nd Workshop on Sponsored Search Auctions, in conjunction with the ACM Conference on Electronic Commerce (EC-06), Ann Arbor, MI, June 2006

Edelsbrunner, H., Guibas, L.J., Pach, J., Pollack, R., Seidel, R., Sharir, M.: Arrangements of curves in the plane: Topology, combinatorics, and algorithms. Theor. Comput. Sci. **92**, 319–336 (1992)

Edlesbrunner, H.: Shape reconstruction with the Delaunay complex. In: LATIN'98, Theoretical Informatics. Lecture Notes in Computer Science, vol. 1380, pp. 119–132. Springer, Berlin (1998)

Edmonds, J.: On the Competitiveness of AIMD-TCP within a General Network. In: LATIN, Latin American Theoretical Informatics, vol. 2976, pp. 577–588 (2004). Submitted to Journal Theoretical Computer Science and/or Lecture Notes in Computer Science

Edmonds, J.: Paths, Trees, and Flowers. Canad. J. Math. **17**, 449–467 (1965)

Edmonds, J.: Scheduling in the dark. Improved results: manuscript 2001. In: Theor. Comput. Sci. **235**, 109–141 (2000). In: 31st Ann. ACM Symp. on Theory of Computing, 1999

Edmonds, J., Chinn, D., Brecht, T., Deng, X.: Non-clairvoyant Multiprocessor Scheduling of Jobs with Changing Execution Characteristics. In: 29th Ann. ACM Symp. on Theory of Computing, 1997, pp. 120–129. Submitted to SIAM J. Comput.

Edmonds, J., Datta, S., Dymond, P.: TCP is Competitive Against a Limited Adversary. In: SPAA, ACM Symp. of Parallelism in Algorithms and Achitectures, 2003, pp. 174–183

Edmonds, J., Pruhs, K.: A maiden analysis of longest wait first. In: Proc. 15th Symp. on Discrete Algorithms (SODA)

Edmonds, J., Pruhs, K.: Multicast pull scheduling: when fairness is fine. Algorithmica **36**, 315–330 (2003)

Edmonds, N., Breuer, A., Gregor, D., Lumsdaine, A.: Single-source shortest paths with the parallel boost graph library. In: 9th DIMACS Implementation Challenge Workshop: Shortest Paths, DIMACS Center, Piscataway, NJ, 13–14 Nov 2006

Efraimidis, P., Spirakis, P.: Weighted Random Sampling with a reservoir. Inf. Process. Lett. J. **97**(5), 181–185 (2006)

Efrat, A., Itai, A., Katz, M.: Geometry Helps in Bottleneck Matching and Related Problems. Algorithmica **31**(1), 1–28 (2001)

Efthymiou, C., Nikoletseas, S., Rolim, J.: Energy Balanced Data Propagation in Wireless Sensor Networks. 4th International Workshop on Algorithms for Wireless, Mobile, Ad-Hoc and Sensor Networks (WMAN '04) IPDPS 2004, Wirel. Netw. J. (WINET) **12**(6), 691–707 (2006)

Efthymiou, C., Nikoletseas, S., Rolim, J.: Energy Balanced Data Propagation in Wireless Sensor Networks. In: Wireless Networks (WINET) Journal, Special Issue on Algorithms for Wireless, Mobile, Ad Hoc and Sensor Networks. Springer (2006)

Efthymiou, C., Spirakis, P.: On the existence of hamiltonian cycles in random intersec- tion graphs. In: Proceedings of 32st International colloquium on Automata, Languages and Programming (ICALP), pp. 690–701. Springer, Berlin Heidelberg (2005)

Efthymiou, C., Spirakis, P.G.: On the Existence of Hamilton Cycles in Random Intersection Graphs. In: Proc. of the 32nd ICALP. LNCS, vol. 3580, pp. 690–701. Springer, Berlin/Heidelberg (2005)

Egecioglu, O., Gonzalez, T.: Minimum-energy Broadcast in Simple Graphs with Limited Node Power. In: Proc. IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS 2001), Anaheim, August 2001 pp. 334–338

Eguchi, A., Fujishige, S., Tamura, A.: A generalized Gale-Shapley algorithm for a discrete-concave stable-marriage model. In: Ibaraki, T., Katoh, N., Ono, H. (eds.) Algorithms and Computation: 14th International Symposium, ISAAC2003. LNCS, vol. 2906, pp. 495–504. Springer, Berlin (2003)

Ekert, A.K.: Quantum cryptography based on Bell's theorem. Phys. Rev. Lett. **67**, 661–663 (1991)

ElGamal, T.: A public-key cryptosystem and a signature scheme based on discrete logarithms. IEEE Trans. Inf. Theor. **31**(4), 469–472 (1985)

Elias, I. Lagergren, J.: Fast Neighbor Joining. In: Proceedings of the 32nd International Colloquium on Automata, Languages, and Programming (ICALP), pp. 1263–1274 (2005)

Elias, I.: Setting the intractability of multiple alignment. In: Proc. of the 14th Annual International Symposium on Algorithms and Computation (ISAAC 2003), 2003, pp. 352–363

Elias, I.: Settling the intractability of multiple alignment. J. Comput. Biol. **13**, 1323–1339 (2006)

Elias, I., Hartman, T.: A 1.375-approximation algorithm for sorting by transpositions. IEEE/ACM Transactions on Computational Biology and Bioinformatics **3**, 369–379 (2006)

Elias, P.: Efficient storage retrieval by content and address of static files. J. ACM, **21**(2):246–260 (1974)

Elias, P.: Error-correcting codes for list decoding. IEEE Trans. Inf. Theory **37**, 5–12 (1991)

Elias, P.: List decoding for noisy channels. Technical Report 335, Research Laboratory of Electronics, MIT, Campridge, MA, USA (1957)

Elias, P., Flower, R.A.: The complexity of some simple retrieval problems. J. Assoc. Comput. Mach. **22**, 367–379 (1975)

Elias, Y., Fernandez, J.M., Mor, T., Weinstein, Y.: Optimal algorithmic cooling of spins. Isr. J. Chem. **46**, 371–391 (2006), also in: Ekl, S. et al. (eds.) Lecture Notes in Computer Science, Volume 4618, pp. 2–26. Springer, Berlin (2007), Unconventional Computation. Proceedings of the Sixth International Conference UC2007 Kingston, August 2007.

Elkin, M.: Computing Almost Shortest Paths. In: Proc. 20th ACM Symp. on Principles of Distributed Computing, Newport, RI, USA, 26–29 Aug. 2001, pp. 53–62

Elkin, M.: Computing Almost Shortest Paths. Trans. Algorithms **1**(2), 283–323 (2005)

Elkin, M., Emek, Y., Spielman, D., Teng, S.-H.: Lower-Stretch Spanning Trees. In: Proc. of the 37th Annual ACM Symp. on Theory of Computing, STOC'05, Baltimore, May 2005, pp. 494–503

Elkin, M., Emek, Y., Spielman, D.A., Teng, S.-H.: Lower-stretch spanning trees. In: STOC '05: Proceedings of the thirty-seventh annual ACM symposium on Theory of computing, pp. 494–503. ACM Press, New York (2005)

Elkin, M., Liebchen, C., Rizzi, R.: New Length Bounds for Cycle Bases. Inf. Proc. Lett. **104**(5), 186–193 (2007)

Elkin, M., Peleg, D.: $(1+\epsilon,\beta)$-Spanner Constructions for General Graphs. SIAM J. Comput. **33**(3), 608–631 (2004)

Elkin, M., Peleg, D.: Spanner constructions for general graphs. In: Proc. of the 33th ACM Symp. on Theory of Computing, Heraklion, 6–8 Jul. 2001, pp. 173–182

Elkin, M., Peleg, D.: Strong inapproximability of the basic k-spanner problem. In: Proc. of 27th International Colloquim on Automata, Languages and Programming, 2000, pp. 636–648

Elkin, M., Zhang, J.: Efficient Algorithms for Constructing $(1+\epsilon,\beta)$-spanners in the Distributed and Streaming Models. Distrib. Comput. **18**(5), 375–385 (2006)

Ellen, F., Fatourou, P., Ruppert, E.: Time lower bounds for implementations of multi-writer snapshots. J. Assoc. Comput. Mach. **54**(6) article 30 (2007)

Elmasri, R., Navanthe, S.B.: Fundamentals of Database Systems, 5th edn. Addison-Wesley, Boston (2007)

Elmasry, A.: On the sequential access theorem and deque conjecture for splay trees. Theor. Comput. Sci. **314**(3), 459–466 (2004)

Emek, Y., Peleg, D.: A tight upper bound on the probabilistic embedding of series-parallel graphs. In: Proc. of Symp. on Discr. Algorithms, SODA'06, Miami, Jan. 2006, pp. 1045–1053

Emerson, E.A.: Temporal and Modal Logic. In: van Leeuwen, J. (ed.) Formal Models and Semantics, vol. B of Handbook of Theoretical Computer Science, pp. 996–1072. Elsevier Science (1990)

Englert, B., Shvartsman, A.A.: Graceful quorum reconfiguration in a robust emulation of shared memory. In: Proc. 20th IEEE International Conference on Distributed Computing Systems, pp. 454–463. Taipei, 10–13 April 2000

Englert, M., Westermann, M.: Lower and upper bounds on FIFO buffer management in qos switches. In: Azar, Y., Erlebach, T. (eds.) Algorithms – ESA 2006, 14th Annual European Symposium, Proceedings. Lecture Notes in Computer Science, vol. 4168, pp. 352–363. Springer, Berlin (2006)

Ephremides, A., Hajek, B.: Information theory and communication networks: an unconsummated union. IEEE Trans. Inf. Theor. **44**, 2416–2434 (1998)

Eppstein, D.: Diameter and treewidth in minor-closed graph families. Algorithmica **27**(3–4), 275–291 (2000)

Eppstein, D.: Dynamic Connectivity in Digital Images. Inf. Process. Lett. **62**(3), 121–126 (1997)

Eppstein, D.: Dynamic Euclidean Minimum Spanning Trees and Extrema of Binary Functions. Discret. Comput. Geom. **13**, 111–122 (1995)

Eppstein, D.: Finding the *k* Shortest Paths. SIAM J. Comput. **28**, 652–673 (1998)

Eppstein, D.: Finding the *k* smallest spanning trees. BIT. **32**, 237–248 (1992)

Eppstein, D.: Geometry in action: minimum spanning trees. http://www.ics.uci.edu/~eppstein/gina/mst.html

Eppstein, D.: Quasiconvex analysis of backtracking algorithms. In: Proceedings of SODA 2004, pp. 781–790

Eppstein, D.: Sequence comparison with mixed convex and concave costs. J. Algorithms **11**(1), 85–101 (1990)

Eppstein, D.: Spanning Trees and Spanners. In: Sack, J.R., Urrutia, J. (eds.) Handbook of Computational Geometry, pp. 425–461. Elsevier, Amsterdam (1999)

Eppstein, D.: Subgraph isomorphism in planar graphs and related problems. J. Graph Algorithms Appl. **3**(3), 1–27 (1999)

Eppstein, D.: The Geometry Junkyard. http://www.ics.uci.edu/~eppstein/junkyard/dilation-free/

Eppstein, D.: Tree-weighted neighbors and geometric *k* smallest spanning trees. Int. J. Comput. Geom. Appl. **4**, 229–238 (1994)

Eppstein, D., Galil, Z., Italiano, G.F., Nissenzweig, A.: Sparsification – a technique for speeding up dynamic graph algorithms. J. Assoc. Comput. Mach. **44**(5), 669–696 (1997)

Eppstein, D., Galil, Z., Italiano, G.F., Spencer, T.H.: Separator based sparsification I: planarity testing and minimum spanning trees. J. Comput. Syst. Sci. Special issue of STOC 93 **52**(1), 3–27 (1996)

Eppstein, D., Galil, Z., Italiano, G.F., Spencer, T.H.: Separator based sparsification II: edge and vertex connectivity. SIAM J. Comput. **28**, 341–381 (1999)

Eppstein, D., Italiano, G.F., Tamassia, R., Tarjan, R.E., Westbrook, J., Yung, M.: Maintenance of a minimum spanning forest in a dynamic plane graph. J. Algorithms **13**, 33–54 (1992)

Eppstein, D., Wortman, K.A.: Minimum Dilation Stars. In: Proc. 21st ACM Symp. Comp. Geom. (SoCG), Pisa, 2005, pp. 321–326

Epstein, L.: A note on on-line scheduling with precedence constraints on identical machines. Inf. Process. Lett. **76**, 149–153 (2000)

Epstein, L., Levin, A.: On the max coloring problem. In: Proc. of the Fifth International Workshop on Approximation and Online Algorithms (WAOA2007) (2007), pp. 142–155

Epstein, L., Levin, A., Woeginger, G.J.: Graph coloring with rejection. In: Proc. of 14th European Symposium on Algorithms (ESA2006), pp. 364–375. (2006)

Epstein, L., Levy, M.: Online interval coloring and variants. In: Proc. of The 32nd International Colloquium on Automata, Languages and Programming (ICALP2005), pp. 602–613. (2005)

Epstein, L., Levy, M.: Online interval coloring with packing constraints. In: Proc. of the 30th International Symposium on Mathematical Foundations of Computer Science (MFCS2005), pp. 295–307. (2005)

Erdong, C., Linji, Y., Hao, Y.: Improved algorithms for 2-interval pattern problem. J. Combin. Optim. **13**(3), 263–275 (2007)

Erdös, P.: Extremal problems in graph theory. In: Theory of Graphs and its Applications (Proc. Sympos. Smolenice, 1963), pp. 29–36. Publ. House Czechoslovak Acad. Sci., Prague (1964)

Erdös, P., Frankl, P., Füredi, Z.: Families of finite sets in which no set is covered by the union of *r* others. Isr. J. Math. **51**, 79–89 (1985)

Erdös, P., Lovász, L.: Problems and results on 3-chromatic hypergraphs and some related questions. Colloq. Math. Soc. János Bolyai **10**, 609–627 (1975)

Erdős, P.L., Steel, M.A., Székely, L. A., Warnow, T.J.: A few logs suffice to build (almost) all trees (II). Theor. Comput. Sci. **221**, 77–118 (1999) Preliminary version as DIMACS TR97-72

Erdős, P.L., Steel, M.A., Székely, L.A., Warnow, T.J.: A few logs suffice to build (almost) all trees (I). Random Struct. Algorithm **14**, 153–184 (1999) Preliminary version as DIMACS TR97-71

Ergun, F., Kumar, R., Rubinfeld, R.: Checking approximate computations of polynomials and functional equations. SIAM J. Comput. **31**(2), 550–576 (2001)

Erlenkotter, D.: A dual-based procedure for uncapacitated facility location problems. Oper. Res. **26**, 992–1009 (1978)

Erlingsson, Ú., Manasse, M., McSherry, F.: A cool and practical alternative to traditional hash tables. In: Proceedings of the 7th Workshop on Distributed Data and Structures (WDAS '06), Santa Clara, CA, USA, 4–6 January 2006

Eskin, E., Halperin, E., Karp, R.: Efficient reconstruction of haplotype structure via perfect phylogeny. J. Bioinform. Comput. Biol. **1**(1), 1–20 (2003)

Estabrook, G.F., Johnson, C.S., Jr., McMorris, F.R.: A mathematical foundation for the analysis of cladistic character compatibility. Math. Biosci. **29**, 181–187 (1976)

Estabrook, G.F., Johnson, C.S., Jr., McMorris, F.R.: An algebraic analysis of cladistic characters. Discret. Math. **16**, 141–147 (1976)

Estivill-Castro, V., Fellows, M.R., Langston, M.A., Rosamond, F.A.: FPT is P-time extremal structure I. In: Algorithms and complexity in Durham 2005. Texts in Algorithmics, vol. 4, pp. 1–41. Kings College Publications, London (2005)

Estrin, D., Govindan, R., Heidemann, J., Kumar, S.: Next Century Challenges: Scalable Coordination in Sensor Networks. In: Proc. 5th ACM/IEEE International Conference on Mobile Computing, MOBICOM'1999

Eswaran, K.P., Gray, J.N., Lorie, R.A., Traiger, I.L.: The notions of consistency and predicate locks in a database system. Commun. ACM **19**(11), 624–633 (1976). doi: http://doi.acm.org/10.1145/360363.360369

Ettinger, M., Høyer, P., Knill, E.: The quantum query complexity of the hidden subgroup problem is polynomial. Inf. Process. Lett. **91**, 43–48 (2004)

Evans, P.A., Smith, A.D., Wareham, H.T.: On the complexity of finding common approximate substrings. Theor. Comput. Sci. **306**(1–3), 407–430 (2003)

Even, G., Naor, J.S., Rao, S., Schieber, B.: Divide-and-conquer approximation algorithms via spreading metrics. J. ACM **47**(4), 585–616 (2000)

Even, S.: Graph Algorithms. Computer Science Press, Potomac (1979)

Even, S., Gazit, H.: Updating distances in dynamic graphs. Method. Oper. Res. **49**, 371–387 (1985)

Even-Dar, E., Kesselman, A., Mansour, Y.: Convergence time to nash equilibria. In: Proc. of the 30th Int. Col. on Aut., Lang. and Progr. (ICALP '03). LNCS, pp. 502–513. Springer, Eindhoven (2003)

Even-Dar, E., Mansour, Y.: Fast convergence of selfish rerouting. In: Proc. of the 16th ACM-SIAM Symp. on Discr. Alg. (SODA '05), SIAM, pp. 772–781. SIAM, Vancouver (2005)

Even-Dar, E., Mansour, Y.: Learning rates for Q-learning. J. Mach. Learn. Res. **5**, 1–25 (2003)

Eyal, E., Halperin, D.: Improved Maintenance of Molecular Surfaces Using Dynamic Graph Connectivity. in: Proc. 5th International Workshop on Algorithms in Bioinformatics (WABI 2005), Mallorca, Spain, 2005, pp. 401–413

Fabri, A., Giezeman, G.-J., Kettner, L., Schirra, S., Schönherr, S.: On the design of CGAL a computational geometry algorithms library. Softw. Pract. Experience **30**(11), 1167–1202 (2000)

Fabri, A., Giezeman, G., Kettner, L., Schirra, S., Schönherr, S.: The cgal kernel: A basis for geometric computation. In: Applied Computational Geometry: Towards Geometric Engineering Proceedings (WACG'96), Philadelphia. Philadelphia, PA, May 27–28, pp. 191–202 (1996)

Fabrikant, A., Papadimitriou, C., Talwar, K.: The complexity of pure nash equilibria. In: Proc. of the 36th ACM Symp. on Th. of Comp. (STOC '04). ACM, Chicago (2004)

Fagerberg, R., Pagh, A., Pagh, R.: External string sorting: Faster and cache-oblivious. In: Proceedings of STACS '06. LNCS, vol. 3884, pp. 68–79. Springer, Marseille (2006)

Faigle, U., Fekete, S., Hochstättler, W., Kern, W.: On the Complexity of Testing Membership in the Core of Min-Cost Spanning Tree Games. Int. J. Game. Theor. **26**, 361–366 (1997)

Faigle, U., Kern, W., Kuipers, J.: Computing the Nucleolus of Min-cost Spanning Tree Games is $\mathcal{NP}$-hard. Int. J. Game Theory **27**, 443–450 (1998)

Faigle, U., Kern, W., Kuipers, J.: On the Computation of the Nucleolus of a Cooperative Game. Int. J. Game Theory **30**, 79–98 (2001)

Faigle, U., Kern, W., Turán, G.: On the performane of online algorithms for partition problems. Acta Cybern. **9**, 107–119 (1989)

Fakcharoenphol, J., Rao, S.: Planar Graphs, Negative Weight Edges, Shortest Paths, and near Linear Time. In: Proc. 42nd IEEE Symp. on Foundations of Computer Science – FOCS (2001), pp. 232–241. IEEE Computer Society Press, Los Alamitos (2001)

Fakcharoenphol, J., Rao, S.: Planar graphs, negative weight edges, shortest paths, and near linear time. J. Comput. Syst. Sci. **72**, 868–889 (2006)

Fakcharoenphol, J., Rao, S., Talwar, K.: A tight bound on approximating arbitrary metrics by tree metrics. In: Proceedings of the 35th annual ACM symposium on Theory of Computing, San Diego, June 2003, pp. 448–455

Fakcharoenphol, J., Rao, S., Talwar, K.: A tight bound on approximating arbitrary metrics by tree metrics. J. Comput. Syst. Sci. **69**, 485–497 (2004)

Fakcharoenphol, J., Rao, S., Talwar, K.: Approximating metrics by tree metrics. SIGACT News **35**, 60–70 (2004)

Fan, R., Lynch, N.A.: Gradient clock synchronization. Distrib. Comput. **18**(4), 255–266 (2006)

Fan, T.-H., Lee, S., Lu, H.-I., Tsou, T.-S., Wang, T.-C., Yao, A.: An optimal algorithm for maximum-sum segment and its application in bioinformatics. Proceedings of the Eighth International Conference on Implementation and Application of Automata. LNCS **2759**, 251–257 (2003)

Fang, F., Blanchette, M.: Footprinter3: phylogenetic footprinting in partially alignable sequences. Nucleic Acids Res. **34**(2), 617–620 (2006)

Fang, Q., Zhu, S., Cai, M., Deng, X.: Membership for core of LP games and other games. COCOON 2001 Lecture Notes in Computer Science, vol. 2108, pp 247–246. Springer-Verlag, Berlin Heidelberg (2001)

Fang, X., Zhu, X., Feng, M., Mao, X., Du, F.: Experimental implementation of dense coding using nuclear magnetic resonance. Phys. Rev. A **61**, 022307 (2000)

Farach, M.: Optimal suffix tree construction with large alphabets. In: Proc. 38th Annu. Symp. Found. Comput. Sci., FOCS 1997, pp. 137–143. IEEE Press, New York (1997)

Farach, M., Muthukrishnan, S.: Optimal parallel dictionary matching and compression. In: Symposium on Parallel Algorithms and Architecture (SPAA), 1995, pp. 244–253

Farach, M., Przytycka, T., Thorup, M.: The maximum agreement subtree problem for binary trees. Proc. of 2nd ESA (1995)

Farach, M., Przytycka, T.M., Thorup, M.: On the agreement of many trees. Inf. Process. Lett. **55**(6), 297–301 (1995)

Farach, M., Thorup, M.: Fast comparison of evolutionary trees. Inf. Comput. **123**(1), 29–37 (1995)

Farach, M., Thorup, M.: Sparse dynamic programming for evolutionary-tree comparison. SIAM J. Comput. **26**(1), 210–230 (1997)

Farach, M., Thorup, M.: String-matching in Lempel–Ziv compressed strings. Algorithmica **20**(4), 388–404 (1998)

Farach-Colton, M., Fernandes, R.J., Mosteiro, M.A.: Bootstrapping a Hop-Optimal Network in the Weak Sensor Model. In: Proc. of the 13th European Symposium on Algorithms (ESA), pp. 827–838 (2005)

Farach-Colton, M., Fernandes, R.J., Mosteiro, M.A.: Lower Bounds for Clear Transmissions in Radio Networks. In: Proc. of the 7th Latin American Symposium on Theoretical Informatics (LATIN), pp. 447–454 (2006)

Farach-Colton, M., Ferragina, P., Muthukrishnan, S.: On the sorting-complexity of suffix tree construction. J. Assoc. Comput. Mach. **47**, 987–1011 (2000)

Farhi, E., Goldstone, J., Gutmann, S.: A quantum algorithm for the Hamiltonian NAND tree. quant-ph/0702144 (2007)

Farhi, E., Goldstone, J., Gutmann, S., Sipser, M.: A limit on the speed of quantum computation in determining parity. Phys. Rev. Lett. **81**, 5442–5444 (1998)

Farhi, E., Gutmann, S.: Quantum computation and decision trees. Phys. Rev. A **58** (1998)

Farris, J.S.: The logical basis of phylogenetic analysis. In: Platnick, N.I., Funk, V.A. (eds.) Advances in Cladistics, pp. 1–36. Columbia Univ. Press, New York (1983)

Farshi, M., Gudmundsson, J.: Experimental study of geometric *t*-spanners. In: Proceedings of the 13th Annual European Symposium on Algorithms. Lect. Notes Comput. Sci. **3669**, 556–567 (2005)

Farzan, A., Ferragina, P., Franceschini, G., Munro, J.I.: Cache-oblivious comparison-based algorithms on multisets. In: Proc. 13th Annual European Symposium on Algorithms. LNCS, vol. 3669, pp. 305–316. Springer, Berlin (2005)

Fatourou, P., Kallimanis, N.D.: Single-scanner multi-writer snapshot implementations are fast! In: Proc. 25th ACM Symposium on Principles of Distrib. Comput. Colorado, July 2006 pp. 228–237. ACM, New York (2006)

Fatourou, P., Mavronicolas, M., Spirakis, P.: Efficiency of oblivious versus non-oblivious schedulers for optimistic, rate-based flow control. SIAM J. Comput. **34**(5), 1216–1252 (2005)

Fatourou, P., Mavronicolas, M., Spirakis, P.: Max-min fair flow control sensitive to priorities. J. Interconnect. Netw. **6**(2), 85–114 (2005) (also in Proceedings of the 2nd International Conference on Principles of Distributed Computing, pp. 45–59 (1998)

Fatourou, P., Mavronicolas, M., Spirakis, P.: The global efficiency of distributed, rate-based flow control algorithms. In: Proceedings of the 5th Colloqium on Structural Information and Communication Complexity, pp. 244–258, June 1998

Feder, T.: A new fixed point approach for stable networks and stable marriages. In: Proceedings of 21st ACM Symposium on Theory of Computing, pp. 513–522, Theory of Computing, Seattle WA, May 1989, pp. 513–522, ACM, New York (1989)

Feder, T.: A new fixed point approach for stable networks and stable marriages. J. Comput. Syst. Sci. **45**, 233–284 (1992)

Feder, T.: Network flow and 2-satisfiability. Algorithmica **11**, 291–319 (1994)

Feder, T.: Stable networks and product graphs. Ph. D. thesis, Stanford University (1991)

Feder, T., Megiddo, N., Plotkin, S.A.: A sublinear parallel algorithm for stable matching. Theor. Comput. Sci. **233**(1–2), 297–308 (2000)

Feder, T., Mihail, M.: Balanced matroids. In: Proceeding 24th ACM Symp. Theory of Computing, pp 26–38, Victoria, British Columbia, Canada, May 04–06 1992

Fedin, S.S., Kulikov, A.S.: Automated proofs of upper bounds on the running time of splitting algorithms. J. Math. Sci. **134**, 2383–2391 (2006). Improved results at http://logic.pdmi.ras.ru/~kulikov/autoproofs.html

Feige, U.: A Threshold of ln *n* for Approximating Set Cover. J. ACM **45**(4) 634–652 (1998)

Feige, U.: Approximating maximum clique by removing subgraphs. SIAM J. Discret. Math. **18**(2), 219–225 (2004)

Feige, U.: Approximating the bandwidth via volume respecting embeddings. J. Comput. Syst. Sci. **60**(3), 510–539 (2000)

Feige, U.: On maximizing welfare when utility functions are subadditive. In: Proc. of the 38th ACM Symposium on Theory of Computing (STOC'06), 2006

Feige, U.: Randomized graph products, chromatic numbers, and the Lovász theta function. Combinatorica **17**(1), 79–90 (1997)

Feige, U.: Relations between average case complexity and approximation complexity. In: 34th Annual ACM Symposium on the Theory of Computing, pp. 534–543, Montréal, May 19–21, 2002

Feige, U., Hajiaghayi, M., Lee, J.R.: Improved approximation algorithms for minimum-weight vertex separators. In: Proceedings of the 37th annual ACM Symposium on Theory of computing (STOC 2005), pp. 563–572. ACM Press, New York (2005)

Feige, U., Kilian, J.: Zero knowledge and the chromatic number. J. Comput. Syst. Sci. **57**, 187–199 (1998)

Feige, U., Krauthgamer, R.: A polylogarithmic approximation of the minimum bisection. SIAM J. Comput. **31**(4), 1090–1118 (2002)

Feige, U., Krauthgamer, R.: A polylogarithmic approximation of the minimum bisection. SIAM Review **48**(1), 99–130 (2006) (Previous versions appeared in Proceedings of 41st FOCS, 1999; and in SIAM J. Comput. 2002)

Feige, U., Langberg, M., Schechtman, G.: Graphs with tiny vector chromatic numbers and huge chromatic numbers. SIAM J. Comput. **33**(6), 1338–1368 (2004)

Feige, U., Mossel, E., Vilenchik, D.: Complete convergence of message passing algorithms for some satisfiability problems. In: Díaz, J., Jansen, K., Rolim, J.D.P., Zwick, U. (eds.) Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 9th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX 2006 and 10th International Workshop on Randomization and Computation, RANDOM 2006, Barcelona, Spain, August 28–30 2006. Lecture Notes in Computer Science, vol. 4110, pp. 339–350. Springer

Feige, U., Raghavan, P.: Exact analysis of hot-potato routing. In: IEEE (ed.) Proceedings of the 33rd Annual, Symposium on Foundations of Computer Science, pp. 553–562, Pittsburgh (1992)

Feige, U., Schechtman, G.: On the optimality of the random hyperplane rounding technique for MAX-CUT. Random Struct. Algorithms **20**(3), 403–440 (2002)

Feige, U., Vilenchik, D.: A local search algorithm for 3-SAT, Tech. rep. The Weizmann Institute, Rehovat, Israel (2004)

Feige, U., Yahalom, O.: On the complexity of finding balanced oneway cuts. Inf. Process. Lett. **87**(1), 1–5 (2003)

Feigenbaum, J., Papadimitriou, C., Sami, R., Shenker, S.: A BGP-based mechanism for lowest-cost routing. In: Proceedings of the 2002 ACM Symposium on Principles of Distributed Computing, pp. 173–182. Monterey, 21–24 July 2002

Feigenbaum, J., Papadimitriou, C.H., Shenker, S.: Sharing the cost of multicast transmissions. J. Comput. Syst. Sci. **63**, 21–41 (2001)

Fekete, S.P., Khuller, S., Klemmstein, M., Raghavachari, B., Young, N.: A network-flow technique for finding low-weightbounded-degree spanning trees. In: Proceedings of the 5th Integer Programming and Combinatorial Optimization Conference (IPCO 1996) and J. Algorithms **24**(2), 310–324 (1997)

Felber, P., Guerraoui, R., Fayad, M.: Putting oo distributed programming to work. Commun. ACM **42**(11), 97–101 (1999)

Feldman, J.: Decoding Error-Correcting Codes via Linear Programming. Ph. D. thesis, Massachusetts Institute of Technology (2003)

Feldman, J., Karger, D.R.: Decoding turbo-like codes via linear programming. In: Proc. 43rd annual IEEE Symposium on Foundations of Computer Science (FOCS), Vancouver, 16–19 November 2002

Feldman, J., Malkin, T., Servedio, R.A., Stein, C., Wainwright, M.J.: LP decoding corrects a constant fraction of errors. In: Proc. IEEE International Symposium on Information Theory, Chicago, 27 June – 2 July 2004

Feldman, J., Stein, C.: LP decoding achieves capacity. In: Symposium on Discrete Algorithms (SODA '05), Vancouver, January (2005)

Feldman, J., Wainwright, M.J., Karger, D.R.: Using linear programming to decode linear codes. In: 37th annual Conf. on Information Sciences and Systems (CISS '03), Baltimore, 12–14 March 2003

Feldman, P.: Optimal Algorithms for Byzantine Agreement. Ph. D. thesis, MIT (1988)

Feldman, P., Micali, S.: An optimal probabilistic protocol for synchronous Byzantine agreement. SIAM J. Comput. **26**(4), 873–933 (1997). Preliminary version in STOC'88

Feldman, V.: Hardness of Approximate Two-level Logic Minimization and PAC Learning with Membership Queries. In: Proceedings of STOC, pp. 363–372 (2006)

Feldman, V.: On attribute efficient and non-adaptive learning of parities and DNF expressions. In: 18th Annual Conference on Learning Theory, pp. 576–590. Springer-Verlag, Berlin Heidelberg (2005)

Feldman, V.: Optimal hardness results for maximizing agreements with monomials. In: Proceedings of Conference on Computational Complexity (CCC), pp. 226–236 (2006)

Feldmann, A., Muthukrishnan, S.: Tradeoffs for packet classification. In: Proc. IEEE INFOCOM, 2000, pp. 1193–1202

Feldmann, R., Gairing, M., Lücking, T., Monien, B., Rode, M.: Nashification and the coordination ratio for a selfish routing game. In: Proc. of the 30th Int. Col. on Aut., Lang. and Progr. (ICALP '03). LNCS, pp. 514–526. Springer, Eindhoven (2003)

Fellows, M.: Blow-ups, win/win's and crown rules: some new directions in FPT. In: Proceedings of the 29th Workshop on Graph Theoretic Concepts in Computer Science (WG 2003). Lecture Notes in Computer Science, vol. 2880, pp. 1–12. Springer, Berlin (2003)

Fellows, M.: Parameterized complexity: the main ideas and some research frontiers. In: Lecture Notes in Computer Science (ISAAC 2001), vol. 2223, pp. 291–307. Springer, Berlin (2001)

Fellows, M., Langston, M.: On well-partial-order theory and its applications to combinatorial problems of VLSI design. SIAM J. Discret. Math. **5**, 117–126 (1992)

Fellows, M., McCartin, C., Rosamond, F., Stege, U.: Coordinatized kernels and catalytic reductions: an improved FPT algorithm for max leaf spanning tree and other problems. In: Proceedings of the 20th Conference on Foundations of Software Technology and Theoretical Computer Science (FST-TCS 2000). Lecture Notes in Theoretical Computer Science 1974, pp. 240–251. Springer, Berlin (2000)

Fellows, M.R.: New Directions and new challenges in algorithm design and complexity, parameterized. In: Lecture Notes in Computer Science, vol. 2748, p. 505–519 (2003)

Fellows, M.R., Gramm, J., Niedermeier, R.: On the parameterized intractability of motif search problems. Combinatorica **26**(2), 141–167 (2006)

Felsenstein, J.: Evolutionary trees from DNA sequences: a maximum likelihood approach. J. Mol. Evol. **17**, 368–376 (1981)

Felsenstein, J.: Inferring Phylogenies. Sinauer Associates, Inc., Sunderland (2004)

Fenwick, P.: Universal codes. In: Sayood, K. (ed.) Lossless Compression Handbook, pp. 55–78, Academic Press, Boston (2003)

Ferguson, D., Yemini, Y., Nikolaou, C.: Microeconomic Algorithms for Load Balancing in Distributed Computer Systems. In: Proceedings of DCS'88, pp. 419–499. San Jose, 13–17 June 1988

Fernandess, Y., Malkhi, D.: On collaborative content distribution using multi-message gossip. In: Twentieth IEEE International Parallel and Distributed Processing Symposium (IPDPS 2006), Greece, April 2006

Fernández, A., Jiménez, E., Raynal, M.: Eventual leader election with weak assumptions on initial knowledge, communication reliability and synchrony. In: Proc International Symposium on Dependable Systems and Networks (DSN), pp. 166–178 (2006)

Fernandez de la Vega, W., Lueker, G.: Bin packing can be solved within $1 + \varepsilon$ in linear time. Combinatorica **1**, 349–355 (1981)

Fernandez, J.M.: De computatione quantica. Dissertation, University of Montreal (2004)

Fernandez, J.M., Lloyd, S., Mor, T., Roychowdhury V.: Practicable algorithmic cooling of spins. Int. J. Quant. Inf. **2**, 461–477 (2004)

Fernández-Baca, D.: The Perfect Phylogeny Problem. In: Cheng, X., Du, D.-Z. (eds.) Steiner Trees in Industry, pp. 203–234. Kluwer Academic Publishers, Dordrecht (2001)

Fernández-Baca, D., Lagergren, J.: A polynomial-time algorithm for near-perfect phylogeny. SIAM J. Comput. **32**, 1115–1127 (2003)

Fernau, H.: Two-layer planarization: improving on parameterized algorithmics. J. Graph Algorithms Appl. **9**, 205–238 (2005)

Fernau, H., Kaufmann, M., Poths, M.: Comparing trees via crossing minimization. In: Ramanujam R., Sen S. (eds.) Foundations of Software Technology and Theoretical Computer Science FSTTCS 2005. LNCS, vol. 3821, pp. 457–469. Springer, Berlin (2005)

Fernholz, D., Ramachandran, V.: The *k*-orientability thresholds for $g_{n,p}$. In: Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '07), pp. 459–468. ACM Press, New Orleans, Louisiana, USA, 7–9 December 2007

Ferragina, P.: Handbook of Computational Molecular Biology. In: Computer and Information Science Series, ch. 35 on "String search in external memory: algorithms and data structures". Chapman & Hall/CRC, Florida (2005)

Ferragina, P., Giancarlo, R., Manzini, G.: The engineering of a compression boosting library: Theory vs practice in bwt compression. In: Proc. 14th European Symposium on Algorithms (ESA). LNCS, vol. 4168, pp. 756–767. Springer, Berlin (2006)

Ferragina, P., Giancarlo, R., Manzini, G.: The myriad virtues of wavelet trees. In: Proc. 33th International Colloquium on Automata and Languages (ICALP), pp. 561–572. LNCS n. 4051. Springer, Berlin, Heidelberg (2006)

Ferragina, P., Giancarlo, R., Manzini, G., Sciortino, M.: Boosting textual compression in optimal linear time. J. ACM **52**, 688–713 (2005)

Ferragina, P., Grossi, R.: Optimal On-Line Search and Sublinear Time Update in String Matching. SIAM J. Comput. **3**, 713–736 (1998)

Ferragina, P., Grossi, R.: The string B-tree: A new data structure for string search in external memory and its applications. J. ACM **46**, 236–280 (1999)

Ferragina, P., Grossi, R., Montangero, M.: A note on updating suffix tree labels. Theor. Comput. Sci. **201**, 249–262 (1998)

Ferragina, P., Luccio, F.: Dynamic dictionary matching in external memory. Inf. Comput. **146**(2), 85–99 (1998)

Ferragina, P., Luccio, F., Manzini, G., Muthukrishnan, S.: Compressing and searching XML data via two zips. In: Proc. 15th World Wide Web Conference (WWW), pp. 751–760. Edingburg, UK(2006)

Ferragina, P., Luccio, F., Manzini, G., Muthukrishnan, S.: Structuring labeled trees for optimal succinctness, and beyond. In: Proc. 46th IEEE Symposium on Foundations of Computer Science (FOCS), pp. 184–193. Cambridge, USA (2005)

Ferragina, P., Manzini, G.: Indexing compressed text. J. ACM **52**, 552–581 (2005)

Ferragina, P., Manzini, G.: Opportunistic data structures with applications. In: Proceedings of Symposium on Foundations of Computer Science, 2000, pp. 390–398

Ferragina, P., Manzini, G., Mäkinen, V., Navarro, G.: Compressed representation of sequences and full-text indexes. ACM Trans. Algorithms **3**(2) Article 20 (2007)

Ferragina, P., Muthukrishnan, S., deBerg, M.: Multi-method dispatching: a geometric approach with applications to string matching. In: Proc. of the Symposium on the Theory of Computing (STOC), 1999, pp. 483–491

Ferragina, P., Venturini, R.: A simple storage scheme for strings achieving entropy bounds. In: Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 690–695. ACM, SIAM (2007)

Ferragina, P., Venturini, R.: A simple storage scheme for strings achieving entropy bounds. Theor. Comput. Sci. **372**, 115–121 (2007)

Fiala, E.R., Greene, D.H.: Data compression with finite windows. Commun. ACM **32**, 490–505 (1989)

Fiat, A., Goldberg, A.V., Hartline, J.D., Karlin, A.R.: Competitive generalized auctions. In: Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC-02), New York, 19–21 May 2002, pp. 72–81. ACM Press, New York (2002)

Fiat, A., Karp, R.M., Luby, M., McGeoch, L.A., Sleator, D.D., Young, N.E.: Competitive paging algorithms. J. Algorithms **12**(4), 685–699 (1991)

Fiat, A., Mendel, M.: Better algorithms for unfair metrical task systems and applications. SIAM J. Comput. **32**, 1403–1422 (2003)

Fiat, A., Naor, M.: Implicit $O(1)$ probe search. SIAM J. Comput. **22**, 1–10 (1993)

Fiat, A., Naor, M., Schmidt, J.P., Siegel, A.: Non-oblivious hashing. J. Assoc. Comput. Mach. **31**, 764–782 (1992)

Fiat, A., Rabani, Y., Ravid, Y.: Competitive *k*-server algorithms. In: Proceedings 31st IEEE Symposium on Foundations of Computer Science (FOCS), pp. 454–463 (1990)

Fiat, A., Ricklin, M.: Competitive algorithms for the weighted server problem. Theor. Comput. Sci. **130**, 85–99 (1994)

Fiat, A., Woeginger, G. (eds.) Online Algorithms – The State of the Art. Springer Lecture Notes in Computer Science, vol. 1442. Springer, Heidelberg (1998)

Fich, F., Luchangco, V., Moir, M., Shavit, N.: Brief announcement: Obstruction-free step complexity: Lock-free DCAS as an example. In: Proc. 19th Annual International Symposium on Distributed Computing, 2005

Fich, F., Luchangco, V., Moir, M., Shavit, N.: Obstruction-free algorithms can be practically wait-free. In: Proc. 19th Annual International Symposium on Distributed Computing, 2005

Fich, F., Ruppert, E.: Hundreds of impossibility results for distributed computing. Distrib. Comput. **16**(2–3), 121–163 (2003)

Fich, F.E.: How hard is it to take a snapshot? In: SOFSEM 2005: Theory and Practice of Computer Science. Liptovský Ján, January 2005, LNCS, vol. 3381, pp. 28–37. Springer (2005)

Fidge, C. J.: Logical time in distributed computing systems. IEEE Comput. **24**, 28–33 (1991)

Fiduccia, C.M., Mattheyses, R.M.: A Linear Time Heuristic for Improving Network Partitions. In: Proc. ACM/IEEE Design Automation Conf., 1982, pp. 175–181

Fill, J.A., Scheinerman, E.R., Singer-Cohen, K.B.: Random intersection graphs when $m = \omega(n)$: an equivalence theorem relating the evolution of the $G(n, m, p)$ and $G(n, p)$ models. Random Struct. Algorithms **16**, 156–176 (2000)

Finden, C.R., Gordon, A.D.: Obtaining common pruned trees. J. Classific. **2**, 255–276 (1985)

Finn G.: Routing and Addressing Problems in Large Metropolitan-scale Internetworks. Tech. Report ISI/RR-87–180, USC/ISI, March (1987)

Finocchi, I., Panconesi, A., Silvestri, R.: An experimental Analysis of Simple Distributed Vertex Coloring Algorithms. Algorithmica **41**, 1–23 (2004)

Fischer, B., Roth, V., Roos, F., Grossmann, J., Baginsky, S., Widmayer, P., Gruissem, W., Buhmann J.: NovoHMM: A Hidden Markov Model for de novo peptide sequencing. Anal. Chem. **77**, 7265–7273 (2005)

Fischer, M.J.: The Consensus Problem in Unreliable Distributed Systems (A Brief Survey). Research Report, YALEU/DCS/RR-273, Yale University, New Heaven (1983)

Fischer, M.J., Lynch, N.A.: A Lower Bound for the Time to Assure Interactive Consistency. Inf. Process. Lett. **14**(4), 183–186 (1982)

Fischer, M.J., Lynch, N.A., Paterson, M.: Impossibility of distributed consensus with one faulty process. In: Proceedings of the 2nd ACM SIGACT-SIGMOD Symposium on Principles of Database System (PODS) Atlante, 21–23 March, pp. 1–7. Association for Computational Machinery (ACM) (1983)

Fischer, M.J., Lynch, N.A., Paterson, M.S.: Impossibility of distributed consensus with one faulty processor. J. ACM **32**(2), 374–382 (1985)

Fischer, S., Vöcking, B.: On the structure and complexity of worst-case equilibria. Theor. Comput. Sci. **378**(2), 165–174 (2007)

Fischer, T.: Optimizing the degree of minimum weight spanning trees, Technical Report TR93–1338. Cornell University, Computer Science Department (1993)

Fischl, B., Sereno, M., Dale, A.: Cortical surface-based analysis II: Inflation, flattening, and a surface-based coordinate system. NeuroImage **9**, 195–207 (1999)

Fishburn, J.P., Dunlop, A. E.: TILOS: A Posynomial Programming Approach to Transistor Sizing. In: Proceedings of the 1985 International Conference on Computer-Aided Design, pp. 326–328. Santa Clara, CA, November 1985

Fishburn, J.P., Schevon, C.A.: Shaping a distributed-RC line to minimize Elmore delay. IEEE Trans. Circuits Syst.-I: Fundam. Theory Appl. **42**(12), 1020–1022 (1995)

Fishburn., J.P.: Shaping a VLSI wire to minimize Elmore delay. In: Proc. European Design and Test Conference pp. 244–251. IEEE Compute Society, Washington D.C. (1997)

Fitch, W.M.: Toward defining the course of evolution: Minimum change for a specified tree topology. Syst. Zool. **20**, 406–416 (1971)

Flammini, M., Klasing, R., Navarra, A., Perennes, S.: Improved approximation results for the minimum energy broadcasting problem. In: Proceedings of the 2004 joint workshop on Foundations of mobile computing (2004)

Flammini, M., Navarra, A., Klasing, R., Pérennes, A.: Improved approximation results for the minimum energy broadcasting problem. DIALM-POMC, pp. 85–91. ACM Press, New York (2004)

Flaxman, A.D.: A spectral technique for random satisfiable 3CNF formulas. In: Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms (Baltimore, MD, 2003), pp. 357–363. ACM, New York (2003)

Fleiner, T.: A Fixed Point Approach to Stable Matchings and Some Applications. Math. Oper. Res. **28**, 103–126 (2003)

Fleiner, T.: A matroid generalization of the stable matching polytope. In: Gerards, B., Aardal K. (eds.) Integer Programming and Combinatorial Optimization: 8th International IPCO Conference. LNCS, vol. 2081, pp. 105–114. Springer, Berlin (2001)

Fleischer, L.: Approximating fractional multicommodity flow independent of the number of commodities. In: Proceedings of the 40th IEEE Symposium on Foundations of Computer Science (FOCS), pp. 24–31, New York, October 1999

Fleischer, L.: Approximating fractional multicommodity flow independent of the number of commodities. SIAM J. Discret. Math. **13**(4), 505–520 (2000)

Fleischer, L., Goemans, M., Mirrokni, V.S., Sviridenko, M.: Tight approximation algorithms for maximum general assignment problems. In: Proceedings of the 16th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA), pp. 611–620 (2006)

Fleischer, L., Könemann, J., Leonardi, S., Schäfer, G.: Simple cost sharing schemes for multicommodity rent-or-buy and stochastic Steiner tree. In: Proc. of the 38th Annual ACM Symposium on Theory of Computing, pp. 663–670. Association for Computing Machinery, New York (2006)

Fleischer, R., Kamphans, T., Klein, R., Langetepe, E., Trippen, G.: Competitive online approximation of the optimal search ratio. In: Proceedings of the 12th European Symposium on Algorithms (ESA'04). Lecture Notes in Computer Science, vol. 3221, pp. 335–346. Springer, Heidelberg (2004)

Fleischer, R., Romanik, K., Schuierer, S., Trippen, G.: Optimal robot localization in trees. Inf. Comput. **171**, 224–247 (2001)

Fleischer, R., Trippen, G.: Experimental studies of graph traversal algorithms. In: Proceedings of the 2nd International Workshop on Experimental and Efficient Algorithms (WEA'03). Lecture Notes in Computer Science, vol. 2647, pp. 120–133. Springer, Heidelberg (2003)

Fleischer, R., Trippen, G.: Exploring an unknown graph efficiently. In: Proceedings of the 13th European Symposium on Algorithms (ESA'05). Lecture Notes in Computer Science, vol. 3669, pp. 11–22. Springer, Heidelberg (2005)

Fleischer, R., Trippen, G.: Optimal robot localization in trees. In: Proceedings of the 16th Annual Symposium on Computational Geometry (SoCG'00), 2000, pp. 373–374. A video shown at the 9th Annual Video Review of Computational Geometry

Fleischer, R., Wahl, M.: On-line scheduling revisited. J. Sched. **3**, 343–353 (2000)

Flocchini, P., Kranakis, E., Krizanc, D., Santoro, N., Sawchuk, C.: Multiple Mobile Agent Rendezvous in the Ring. In: Proc. LATIN 2004. LNCS, vol. 2976, pp. 599–608. Bueons Aires, 5–8 April 2004

Flum, J., Grohe, M.: Parameterized Complexity Theory. Texts in Theoretical Computer Science, vol. XIV. An EATCS Series. Springer, Berlin (2006)

Flum, J., Grohe, M.: The Parameterized complexity of counting problems. SIAM J. Comput. **33**(4), 892–922 (2004)

Flury, R., Wattenhofer, R.: MLS: An Efficient Location Service for Mobile Ad Hoc Networks. In: Proceedings of the 7th ACM Int. Symposium on Mobile Ad-Hoc Networking and Computing (MobiHoc), Florence, Italy, May 2006

Fomin, F., Grandoni, F., Kratsch, D.: Measure and conquer: a simple $O(2^{0.288n})$ independent set algorithm. In: Proc. 17th Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA 2006), pp. 18–25 (2006)

Fomin, F.V., Gaspers, S., Pyatkin, A.V.: Finding a minimum feedback vertex set in time $O(1.7548^n)$. In: Proc. 2th IWPEC. LNCS, vol. 4196, pp. 184–191. Springer, Berlin (2006)

Fomin, F.V., Grandoni, F., Kratsch, D.: Measure and conquer: Domination – A case study. In: Proceedings of ICALP 2005. LNCS, vol. 3380, pp. 192–203. Springer, Berlin (2005)

Fomin, F.V., Kratsch, D., Todinca, I.: Exact (exponential) algorithms for treewidth and minimum fill-in. In: ICALP of LNCS, vol. 3142, pp. 568–580. Springer, Berlin (2004)

Fomin, F.V., Kratsch, D., Todinca, I., Villanger, I.: Exact (exponential) algorithms for treewidth and minimum fill-in (2006). To appear in SIAM Journal of Computing, Preliminary version appeared in ICALP 2004

Fomin, F.V., Kratsch, D., Woeginger, G.J.: Exact (exponential) algorithms for the dominating set problem. In: Proceedings of WG 2004. LNCS, vol. 3353, pp. 245–256. Springer, Berlin (2004)

Fomin, F.V., Mazoit, F., Todinca, I.: Computing branchwidth via efficient triangulations and blocks. In: Proceedings of the 31st Workshop on Graph Theoretic Concepts in Computer Science (WG 2005). Lecture Notes Computer Science, vol. 3787, pp. 374–384. Springer, Berlin (2005)

Fomin, F.V., Thilikos, D.M.: Dominating sets in planar graphs: Branch-width and exponential speed-up. SIAM J. Comput. **36**, 281–309 (2006)

Fomin, F.V., Thilikos, D.M.: Fast parameterized algorithms for graphs on surfaces: Linear kernel and exponential speed-up. In: Proceedings of the 31st International Colloquium on Automata, Languages and Programming (ICALP 2004). Lecture Notes Computer Science, vol. 3142, pp. 581–592. Springer, Berlin (2004)

Fomin, F.V., Thilikos, D.M.: New upper bounds on the decomposability of planar graphs. J. Graph Theor. **51**, 53–81 (2006)

Fonseca, R., Ratnasamy, S., Zhao, J., Ee, C.T., Culler, D., Shenker, S., Stoica, I.: Beacon Vector Routing: Scalable Point-to-Point Routing in Wireless Sensornets. In: 2nd Symposium on Networked Systems Design & Implementation (NSDI), Boston, Massachusetts, USA, May 2005

Ford, L.R., Fulkerson, D.R.: Maximal flow through a network. Can. J. Math. **8**, 399–404. (1956)

Fortune, S.J.: Stable maintenance of point-set triangulations in two dimensions. IEEE Found. Comput. Sci.: **30**, 494–499 (1989)

Fortune, S.J., van Wyk, C.J.: Efficient exact arithmetic for computational geometry. In: Proceeding 9th ACM Symposium on Computational Geometry, pp. 163–172 (1993)

Foschini, L., Grossi, R., Gupta, A., Vitter, J.S.: When indexing equals compression: Experiments with compressing suffix arrays and applications. ACM Trans. Algorithms **2**(4), 611–639 (2006)

Fotakis, D.: On the competitive ratio for online facility location. In: Proceedings of the 30th International Colloquium on Automata, Languages and Programming (ICALP). Lecture Notes in Computer Science, vol. 2719, pp. 637–652. Springer, Berlin (2003)

Fotakis, D., Kontogiannis, S., Koutsoupias, E., Mavronicolas, M., Spirakis, P.: The structure and complexity of nash equilibria for a selfish routing game. In: Proc. of the 29th Int. Col. on Aut., Lang. and Progr. (ICALP '02). LNCS, pp. 123–134. Springer, Málaga (2002)

Fotakis, D., Kontogiannis, S., Spirakis, P.: Atomic congestion games among coalitions. In: Proc. of the 33rd Int. Col. on Aut., Lang. and Progr. (ICALP '06). LNCS, vol. 4051, pp. 572–583. Springer, Venice (2006)

Fotakis, D., Kontogiannis, S., Spirakis, P.: Selfish unsplittable flows. J. Theoret. Comput. Sci. **348**, 226–239 (2005)

Fotakis, D., Kontogiannis, S., Spirakis, P.: Symmetry in Network Congestion Games: Pure Equilibria and Anarchy Cost. In: Proc. of the 3rd Workshop of Approximate and On-line Algorithms (WAOA 2005). Lecture Notes in Computer Science (LNCS), vol. 3879, pp. 161–175. Springer, Berlin Heidelberg (2006)

Fotakis, D., Nikoletseas, S., Papadopoulou, V., Spirakis, P.: $\mathcal{NP}$-Completeness Results and Efficient Approximations for Radiocoloring in Planar Graphs. In: Proceedings of the 25th International Symposium on Mathematical Foundations of Computer Science, Lecture Notes of Computer Science, vol. 1893, pp. 363–372. Springer (2000)

Fotakis, D., Nikoletseas, S., Papadopoulou, V.G., Spirakis, P.G.: Radiocoloring in Planar Graphs: Complexity and Approximations. Theor. Comput. Sci. Elsevier **340**, 514–538 (2005)

Fotakis, D., Pagh, R., Sanders, P., Spirakis, P.G.: Space efficient hash tables with worst case constant access time. Theor. Comput. Syst. **38**(2), 229–248 (2005)

Fotakis, D., Pantziou, G., Pentaris, G., Spirakis, P.: Frequency Assignment in Mobile and Radio Networks. In: Networks in Distributed Computing, DIMACS Series in Discrete Mathematics and Theoretical Computer Science 45, pp. 73–90 (1999)

Fotakis, D., Spirakis, P.: Minimum Congestion Redundant Assignments to Tolerate Random Faults. Algorithmica **32**, 396–422 (2002)

Fowler, R.J., Paterson, M.S., Tanimoto, S.L.: Optimal packing and covering in the plane are NP-complete. Inf. Process. Lett. **12**(3), 133–137 (1981)

Fraenkel, A.S., Klein, S.T.: Novel compression of sparse bit-strings – Preliminary report. In: Apostolico, A., Galil, Z. (eds) Combinatorial Algorithms on Words, NATO ASI Series F, vol. 12, pp. 169–183. Springer, Berlin (1985)

Fraenkel, A.S., Simpson, R.J.: How many squares can a string contain? J. Comb. Theory Ser. A **82**, 112–120 (1998)

Fraenkel, A.S., Simpson, R.J.: The Exact Number of Squares in Fibonacci Words. Theor. Comput. Sci. **218**(1), 95–106 (1999)

Fraigniaud, P., Gauron, P.: The content-addressable network D2B. Tech. Report 1349, LRI, Univ. Paris-Sud (2003)

Fraigniaud, P., Ilcinkas, D., Peer, G., Pelc, A., Peleg, D.: Graph exploration by a finite automaton. Theor. Comput. Sci. **345**, 331–344 (2005)

Frances, M., Litman, A.: On covering problems of codes. Theor. Comput. Syst. **30**, 113–119 (1997)

Franceschini, G.: Proximity mergesort: Optimal in-place sorting in the cache-oblivious model. In: Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), SIAM, p. 291. Philadelphia, 2004

Franceschini, G., Grossi, R.: A general technique for managing strings in comparison-driven data structures. In: Annual International Colloquium on Automata, Languages and Programming (ICALP), 2004

Franceschini, G., Grossi, R.: Optimal in-place sorting of vectors and records. In: Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP '05). LNCS, vol. 3580, pp. 90–102. Springer, Lisbon (2005)

Franceschini, G., Grossi, R.: Optimal worst-case operations for implicit cache-oblivious search trees. In: Proc. Algorithms and Data Structures, 8th International Workshop, WADS. LNCS, vol. 2748, pp. 114–126. Springer, Berlin (2003)

Francis, Y.L., Chin, N.L.H., Lam, T.W., Prudence, W.H.W.: Efficient constrained multiple sequence alignment with performance guarantee. J. Bioinform. Comput. Biol. **3**(1), 1–18 (2005)

Franco, J.: Probabilistic analysis of the pure literal heuristic for the satisfiability problem. Annal. Oper. Res. **1**, 273–289 (1984)

Franco, J.: Results related to threshold phenomena research in satisfiability: Lower bounds. Theor. Comput. Sci. **265**, 147–157 (2001)

Franek, F., Karaman, A., Smyth, W.F.: Repetitions in Sturmian strings. Theor. Comput. Sci. **249**(2), 289–303 (2000)

Franek, F., Simpson, R.J. , and Smyth, W.F.: The maximum number of runs in a string. In: Proc. 14-th Australian Workshop on Combinatorial Algorithms, pp. 26–35. Curtin University Press, Perth (2003)

Franek, F., Smyth, W.F., Tang, Y.: Computing all repeats using suffix arrays. J. Autom. Lang. Comb. **8**(4), 579–591 (2003)

Frank, A.: Packing paths, cuts, and circuits – a survey. In: Korte, B., Lovász, L., Prömel H.J., Schrijver A. (eds.) Paths, Flows and VLSI-Layout, pp. 49–100. Springer, Berlin (1990)

Frank, A., Pevzner, P.: Pepnovo: De novo peptide sequencing via probabilistic network modeling. Anal. Chem. **77**, 964–973 (2005)

Fraser, K., Harris, T.: Concurrent programming without locks. http://www.cl.cam.ac.uk/netos/papers/2004-cpwl-submission.pdf (2004)

Frederickson, G., Lynch, N.: The impact of synchronous communication on the problem of electing a leader in a ring. In: Proc. of the 16th Annual ACM Symposium on Theory of Computing, pp. 493–503. ACM, USA (1984)

Frederickson, G.N.: A data structure for dynamically maintaining rooted trees. J. Algorithms **24**(1), 37–65 (1997)

Frederickson, G.N.: Ambivalent data structures for dynamic 2-edge-connectivity and k smallest spanning trees. SIAM J. Comput. **26**(2), 484–538 (1997)

Frederickson, G.N.: Data structures for on-line update of minimum spanning trees, with applications. SIAM J. Comput. **14**(4), 781–798 (1985)

Frederickson, G.N., JáJá, J.: On the relationship between the biconnectivity augmentation and Traveling Salesman Problem. Theor. Comput. Sci. **19**(2), 189–201 (1982)

Frederickson, G.N., Srinivas, M.A.: Algorithms and data structures for an expanded family of matroid intersection problems. SIAM. J. Comput. **18**, 112–138 (1989)

Fredman, M., Tarjan, R.: Fibonacci heaps and their uses in improved network optimization algorithms. J. ACM **34**(3), 596–615 (1987)

Fredman, M.L.: New bounds on the complexity of the shortest path problem. SIAM J. Comput. **5**(1), 83–89 (1976)

Fredman, M.L.: New bounds on the complexity of the shortest path problems. SIAM J. Comp. **5**(1), 87–89 (1976)

Fredman, M.L.: Two applications of a probabilistic search technique: sorting $X + Y$ and building balanced search trees. Proc. of the 7th ACM STOC, pp. 240–244 (1975)

Fredman, M.L., Komlós, J., Szemerédi, E.: Storing a sparse table with $O(1)$ worst case access time. J. Assoc. Comput. Mach. **31**(3), 538–544 (1984)

Fredman, M.L., Saks, M.E.: The cell probe complexity of dynamic data structures. In: Proc. 21st ACM Symposium on Theory of Computing (STOC), 1989, pp. 345–354

Fredman, M.L., Willard, D.E.: Surpassing the information theoretic bound with fusion trees. J. Comput. Syst. Sci. **47**(3), 424–436 (1993). See also STOC'90

Fredman, M.L., Willard, D.E.: Trans-dichotomous algorithms for minimum spanning trees and shortest paths. J. Comput. Syst. Sci. **48**(3), 533–551 (1994). See also FOCS'90

Fredriksson, K., Grabowski, S.: Practical and optimal string matching. In: Proceedings of SPIRE'2005. LNCS, vol. 3772, pp. 374–385. Springer, Berlin (2005)

Fredriksson, K., Mäkinen, V., Navarro, G.: Rotation and lighting invariant template matching. In: Proceedings of the 6th Latin American Symposium on Theoretical Informatics (LATIN'04). LNCS, pp. 39–48 (2004)

Fredriksson, K., Mozgovoy, M.: Efficient parameterized string matching. Inf. Process. Lett. **100**(3), 91–96 (2006)

Fredriksson, K., Navarro, G.: Average-optimal single and multiple approximate string matching. ACM J. Exp. Algorithms **9**(1.4) (2004)

Fredriksson, K., Navarro, G., Ukkonen, E.: Faster than FFT: Rotation invariant combinatorial template matching. In: Pandalai, S. (ed.) Recent Research Developments in Pattern Recognition, vol. II, pp. 75–112. Transworld Research Network, Trivandrum, India (2002)

Fredriksson, K., Navarro, G., Ukkonen, E.: Optimal exact and fast approximate two dimensional pattern matching allowing rotations. In: Proceedings of the 13th Annual Symposium on Combinatorial Pattern Matching (CPM 2002). LNCS, vol. 2373, pp. 235–248 (2002)

Fredriksson, K., Navarro, G., Ukkonen, E.: Sequential and Indexed Two-Dimensional Combinatorial Template Matching Allowing Rotations. Theor. Comput. Sci. **347**(1–2), 239–275 (2005)

Fredriksson, K., Ukkonen, E.: A rotation invariant filter for two-dimensional string matching. In: Proc. 9th Annual Symposium on Combinatorial Pattern Matching (CPM). LNCS, vol. 1448, pp. 118–125. Springer, Berlin (1998)

Fredriksson, K., Ukkonen, E.: Combinatorial methods for approximate pattern matching under rotations and translations in 3D arrays. In: Proc. 7th International Symposium on String Processing and Information Retrieval, pp. 96–104. IEEE Computer Society, Washington, DC (2000)

Freedman, M.: P/NP and the quantum field computer. Proc. Natl. Acad. Sci. USA **95**, 98–101 (1998)

Freedman, M., Kitaev, A., Larsen, M., Wang, Z.: Topological quantum computation. Mathematical challenges of the 21st century. (Los Angeles, CA, 2000). Bull. Amer. Math. Soc. (N.S.) **40**(1), 31–38 (2003)

Freedman, M.H., Kitaev, A., Wang, Z.: A modular Functor which is universal for quantum computation. Commun. Math. Phys. **227**(3), 605–622 (2002)

Freedman, M.H., Kitaev, A., Wang, Z.: Simulation of topological field theories by quantum computers. Commun. Math. Phys. **227**, 587–603 (2002)

Freedman, M.J., Freudenthal, E., Mazières, D.: Democratizing content publication with coral. In: Proceedings of the 1st USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI '04), March 2004

Freedman, M.J., Mazières, D.: Sloppy hashing and self-organizing clusters. In: Proceedings of the 2nd Intl. Workshop on Peer-to-Peer Systems (IPTPS '03), February 2003

Freivalds, R.: Fast probabilistic algorithms. In: Proceedings of the 8th Symposium on Mathematical Foundations of Computer Science, pp. 57–69, Olomouc, Czechoslovakia, 3–7 September 1979

Freivalds, R., Kinber, E., Smith, C.H.: On the Intrinsic Complexity of Learning. Inform. Comput. **118**(2), 208–226 (1995)

Freivalds, R., Kinber, E., Wiehagen, R.: How inductive inference strategies discover their errors. Inform. Comput. **123**(1), 64–71 (1995)

Freivalds, R., Smith, C.H.: On the Role of Procrastination in Machine Learning. Inform. Comput. **107**(2), 237–271 (1993)

Freund, A., Karloff, H.: A lower bound of $8/(7+\frac{1}{k-1})$ on the integrality ratio of the Calinescu–Karloff–Rabani relaxation for Multiway Cut. Inf. Process. Lett. **75**, 43–50 (2000)

Freund, Y., Schapire, R. E.: Large margin classification using the perceptron algorithm. In: Proceedings of the Eleventh Annual Conference on Computational Learning Theory (1998)

Frick, M., Grohe, M.: Deciding first-order properties of locally tree-decomposable structures. J. ACM **48**(6), 1184–1206 (2001)

Friedgut, E.: Sharp thresholds of graph properties, and the *k*-sat problem. J. AMS **12**, 1017–1054 (1997)

Frieze, A., Jerrum, M.R.: Improved approximation algorithms for MAX-*k*-CUT and MAX BISECTION. Algorithmica **18**, 61–77 (1997)

Frieze, A., Kannan, R.: The Regularity Lemma and Approximation Schemes for Dense Problems. In: Proc. 37th IEEE FOCS 1996, pp. 12–20. IEEE Computer Society Press, Los Alamitos

Frigioni, D., Marchetti-Spaccamela, A., Nanni, U.: Fully dynamic algorithms for maintaining shortest paths trees. J. Algorithm **34**, 351–381 (2000)

Frigioni, D., Marchetti-Spaccamela, A., Nanni, U.: Semi-dynamic algorithms for maintaining single source shortest paths trees. Algorithmica **22**, 250–274 (1998)

Frigioni, D., Miller, T., Nanni, U., Zaroliagis, C.D.: An experimental study of dynamic algorithms for transitive closure. ACM J Exp. Algorithms **6**(9) (2001)

Frigo, M., Johnson, S. G.: FFTW: An adaptive software architecture for the FFT. In: Proc. IEEE Int'l Conf. Acoustics, Speech, and Signal Processing, vol. 3, pp. 1381–1384, Seattle, WA (1998)

Frigo, M., Leiserson, C.E., Prokop, H., Ramachandran, S.: Cache-oblivious algorithms. In: 40th Annual IEEE Symposium on Foundations of Computer Science, pp. 285–298. IEEE Computer Society Press, Los Alamitos (1999)

Ftp site of DIMACS implementation challenges, ftp://dimacs.rutgers.edu/pub/challenge/

Fuchs, B.: On the hardness of range assignment problems. In: Proceedings of the 6th Italian Conference on Algorithms and Complexity (CIAC), pp. 127–138 (2006)

Fuchs, C.A., Gisin, N., Griffiths, R.B., Niu, C., Peres, A.: Optimal eavesdropping in quantum cryptography, I. Information bound and optimal strategy. Phys. Rev. A **56**, 1163–1172 (1997)

Fujii, M., Kasami, T., Ninomiya, K.: Optimal sequencing of two equivalent processors. SIAM J. Comput. **17**, 784–789 (1969)

Fujishige, S., Tamura, A.: A Two-Sided Discrete-Concave Market with Bounded Side Payments: An Approach by Discrete Convex Analysis. Math. Oper. Res. **32**, 136–155 (2007)

Fukuda, T., Morimoto, Y., Morishita, S., Tokuyama, T.: Mining Optimized Association Rules for Numeric Attributes. J. Comput. Syst. Sci. **58**, 1–12 (1999)

Fulkerson, D.R., Gross, O.A.: Incidence matrices and interval graphs. Pac. J. Math. **15**(3), 835–855 (1965)

Füredi, Z.: On *r*-cover-free families. J. Comb. Theory, Series A **73**, 172–173 (1996)

Fürer, M., Raghavachari, B.: An NC approximation algorithm for the minimum-degree spanning tree problem. In: Proceedings of the 28th Annual Allerton Conference on Communication, Control and Computing, 1990, pp. 174–281

Fürer, M., Raghavachari, B.: Approximating the minimum degree spanning tree to within one from the optimal degree. In: Proceedings of the Third Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 1992), 1992, pp. 317–324

Fürer, M., Raghavachari, B.: Approximating the minimum-degree Steiner tree to within one of optimal. J. Algorithms **17**(3), 409–423 (1994)

Furst, M., Jackson, J., Smith, S.: Improved learning of AC^0 functions. In: Proceedings of the Fourth Annual Workshop on Computational Learning Theory, pp. 317–325, Santa Cruz, (1991)

Furst, M., Saxe, J., Sipser, M.: Parity, circuits, and the polynomial time hierarchy. Math. Syst. Theor. **17**(1), 13–27 (1984)

Gabow, H.: Data structures for weighted matching and nearest common ancestors with linking. In: Symp. on Discrete Algorithms, 1990, pp. 434–443

Gabow, H.N.: A matroid approach to finding edge connectivity and packing arborescences. J. Comput. Syst. Sci. **50**, 259–273 (1995)

Gabow, H.N.: An almost linear time algorithm for two processors scheduling. J. ACM **29**(3), 766–780 (1982)

Gabow, H.N.: An ear decomposition approach to approximating the smallest 3-edge connected spanning subgraph of a multigraph. SIAM J. Discret. Math. **18**(1), 41–70 (2004)

Gabow, H.N.: Better performance bounds for finding the smallest k-edge connected spanning subgraph of a multigraph. In: SODA, 2003, pp. 460–469

Gabow, H.N., Bentley, J.L., Tarjan, R.E.: Scaling and Related Techniques for Geometry Problems. In: STOC 1984, pp. 135–143

Gabow, H.N., Galil, Z., Spencer, T.H., Tarjan, R.E.: Efficient algorithms for finding minimum spanning trees in undirected and directed graphs. Combinatorica **6**, 109–122 (1986)

Gabow, H.N., Goemans, M.X., Williamson, D.P.: An efficient approximation algorithm for the survivable network design problem. Math. Program. Ser. B **82**(1–2), 13–40 (1998)

Gabow, H.N., Tarjan, R.E.: Faster scaling algorithms for general graph matching problems. J. ACM **38**(4), 815–853 (1991)

Gabriel, K.R., Sokal, R.R.: A new statistical approach to geographic variation analysis. Syst. Zool. **18**, 259–278 (1969)

Gafni, E., Bertsekas, D.: Dynamic control of session input rates in communication networks. IEEE Trans. Autom. Control **29**(11), 1009–1016 (1984)

Gafni, E., Bertsekas, D.P.: Distributed algorithms for generating loop-free routes in networks with frequently changing topology. IEEE Trans. Commun. **29**(1), 11–18 (1981)

Gafni, E., Guerraoui, R., Pochon, B.: From a Static Impossibility to an Adaptive Lower Bound: The Complexity of Early Deciding Set Agreement. In: Proc. 37th ACM Symposium on Theory of Computing (STOC 2005), pp. 714–722. ACM Press, New York (2005)

Gafni, E., Koutsoupias, E.: Three-processor tasks are undecidable. SIAM J. Comput. **28**(3), 970–983 (1999)

Gafni, E., Rajsbaum, S., Herlihy, M.: Subconsensus Tasks: Renaming is Weaker than Set Agreement. In: Proc. 20th Int'l Symposium on Distributed Computing (DISC'06). LNCS, vol. 4167, pp. 329–338. Springer, Berlin (2006)

Gairing, M., Lücking, T., Mavronicolas, M., Monien, B.: The price of anarchy for polynomial social cost. Theor. Comput. Sci. **369**(1-3), 116–135 (2006)

Gairing, M., Luecking, T., Mavronicolas, M., Monien, B.: The price of anarchy for restricted parallel links. Parallel Process. Lett. **16**, 117–131 (2006) Preliminary version appeared in STOC 2004

Gairing, M., Monien, B., Tiemann, K.: Routing (un-)splittable flow in games with player specific linear latency functions. In: Proc. of the 33rd Int. Col. on Aut., Lang. and Progr. (ICALP '06). LNCS, pp. 501–512. Springer, Venice (2006)

Gal, S.: Minimax solutions for linear search problems. SIAM J. Appl. Math. **27**, 17–30 (1974)

Gal, S.: Search Games, pp. 109–115, 137–151, 189–195. Academic Press, New York (1980)

Galambos, G., Woeginger, G.J.: Online bin packing – a restricted survey. ZOR Math. Methods Oper. Res. **42**, 25–45 (1995)

Gale, D., Shapley, L.S.: College admissions and the stability of marriage. Am. Math. Mon. **69**, 9–15 (1962)

Gale, D., Sotomayor, M.: Some remarks on the stable matching problem. Discret. Appl. Math. **11**, 223–232 (1985)

Galil, Z., Giancarlo, R.: Speeding up dynamic programming with applications to molecular biology. Theor. Comput. Sci. **64**, 107–118 (1989)

Galil, Z., Italiano, G. F.: Fully dynamic algorithms for 2-edge-connectivity. SIAM J. Comput. **21**, 1047–1069 (1992)

Galil, Z., Italiano, G.F.: Maintaining the 3-edge-connected components of a graph on-line. SIAM J. Comput. **22**, 11–28 (1993)

Galil, Z., Italiano, G.F., Sarnak, N.: Fully dynamic planarity testing with applications. J. ACM **48**, 28–91 (1999)

Galil, Z., Park, J.G., Park, K.: Three-dimensional periodicity and its application to pattern matching. SIAM J. Discret. Math. **18**, 362–381 (2004)

Galil, Z., Park, K.: Alphabet-independent two-dimensional witness computation. SIAM. J. Comput. **25**(5), 907–935 (1996)

Galil, Z., Seiferas, J.: Time-space optimal string matching. J. Comput. Syst. Sci. **26**(3), 280–294 (1983)

Gallager, R.: A perspective on multiaccess communications. IEEE Trans. Inf. Theor. **31**, 124–142 (1985)

Gallager, R.: Low-density parity-check codes. IRE Trans. Inform. Theory, IT-8 , pp. 21–28 (1962)

Gallager, R.G., Humblet, P.A., Spira, P.M.: A distributed algorithm for minimum-weight spanning trees. ACM Trans. Prog. Lang. Systems **5**(1), 66–77 (1983)

Ganapathy, G., Warnow, T.J.: Approximating the complement of the maximum compatible subset of leaves of *k* trees. In: Proc. of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX'02), LCNS, vol. 2462, pp. 122–134., Springer, Berlin (2002)

Ganapathy, G., Warnow, T.J.: Finding a maximum compatible tree for a bounded number of trees with bounded degree is solvable in polynomial time. In: Gascuel, O., Moret, B.M.E. (eds.) Proc. of the 1st International Workshop on Algorithms in Bioinformatics (WABI'01), pp. 156–163 (2001)

Gandhi, R., Khuller, S., Parthasarathy, S., Srinivasan, A.: Dependent rounding and its applications to approximation algorithms. J. ACM **53**(3), 324–360 (2006)

Gandhi, R., Parthasarathy, S.: Distributed Algorithms for Coloring and Connected Domination in Wireless Ad Hoc Networks. In: Foundations of Software Technology and Theoretical Computer Science (FSTTCS), pp. 447–459 (2004)

Gao, J., Guibas, L., Hershberger, J., Zhang, L., Zhu, A.: Geometric Spanner for Routing in Mobile Networks. In: Proc. 2nd ACM Int. Symposium on Mobile Ad-Hoc Networking and Computing (MobiHoc), Long Beach, CA, USA, October 2001

Gao, J., Guibas, L.J., Hershberger, J., Zhang, L., Zhu, A.: Discrete mobile centers. Discrete Comput. Geom. **30**, 45–63 (2003)

Gao, J., Guibas, L.J., Hershberger, J., Zhang, L., Zhu, A.: Geometric spanners for routing in mobile networks. IEEE J. Sel. Areas Commun. Wirel. Ad Hoc Netw. (J-SAC), **23**(1), 174–185 (2005)

Gao, J., Guibas, L.J., Nguyen, A.: Deformable spanners and applications. In: Proceedings of the 20th ACM Symposium on Computational Geometry, pp. 190–199, New York, 9–11 June 2004

Gao, J., Zhang, L.: Well-separated pair decomposition for the unit-disk graph metric and its applications. In: Proc. of 35th ACM Symposium on Theory of Computing (STOC'03), 2003, pp. 483–492

Gao, Y., Wong, D.F.: Wire-sizing for delay minimization and ringing control using transmission line model. In: Proc. Conf. on Design Automation and Test in Europe, pp. 512–516. ACM, New York (2000)

Garay, J.A., Moses, Y.: Fully Polynomial Byzantine Agreement for n > 3t Processors in t + 1 Rounds. SIAM J. Comput. **27**(1), 247–290 (1998)

Garcia-Molina, H., Barbara, D.: How to assign votes in a distributed system. J. ACM **32**, 841–860 (1985)

García, Y.J., López, M.A., Leutenegger, S.T.: A greedy algorithm for bulk loading R-trees. In: Proc. 6th ACM Symposium on Advances in GIS, 1998, pp. 163–164

Gardener, M.K.: Probabilistic Analysis and Scheduling of Critical Soft Real-Time Systems. Ph. D. thesis, Department of Computer Science, University of Illinois at Urbana-Champaign (1999)

Gardenfors, P.: Match Making: assignments based on bilateral preferences. Behav. Sci. **20**, 166–173 (1975)

Gardner, P.P., Giegerich, R.: A comprehensive comparison of comparative RNA structure prediction approaches. BMC Bioinformatics **30**, 140 (2004)

Garey, M., Johnson, D., Stockmeyer, L.: Some simplified NP-complete graph problems. Theor. Comput. Sci. **1**, 237–267 (1976)

Garey, M., Johnson, D.S.: Computers and Intractability. W. H. Freeman, San Francisco (1979)

Garey, M.R., Graham, R.L., Johnson, D.S.: Some NP-complete geometric problems. In: Proceedings of 8th Annual ACM Symposium on Theory of Computing (STOC '76), pp. 10–22. Association for Computing Machinery, New York (1976)

Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, New York (1979)

Garg, N., Könemann., J.: Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In: Proceedings of the 39th IEEE Symposium on Foundations of Computer Science (FOCS), pp. 300–309. (1998)

Garg, N., Kumar, A.: Better algorithms for minimizing average flowtime on related machines. In: Proceesings of ICALP, pp. 181–190 (2006)

Garg, N., Kumar, A.: Minimizing average flow time on related machines. In: ACM Symposium on Theory of Compuring (STOC), pp. 730–738 (2006)

Garg, N., Vazirani, V., Yannakakis, M.: Primal-Dual Approximation Algorithms for Integral Flow and Multicut in Trees. Algorithmica **18**(1), 3–20 (1997). Preliminary version appeared in Proc. ICALP 1993

Garg, N., Vazirani, V.V., Yannakakis, M.: Approximate max-flow min-(multi)cut theorems and their applications. SIAM Comput. J., **25**(2), 235–251. (1996)

Garg, N., Vazirani, V.V., Yannakakis, M.: Multiway cuts in node weighted graphs. J. Algorithms **50**(1), 49–61 (2004). Preliminary version in ICALP 1994

Garg, N., Vempala, S., Singla, A.: Improved approximation algorithms for biconnected subgraphs via better lower bounding techniques. In: SODA, 1993, pp. 103–111

Garg, R., Kapoor, S.: Auction algorithms for market equilibrium, In: Proceedings of STOC'04, pp. 511–518. ACM, Chicago (2004)

Garnerone, S., Marzuoli, A., Rasetti, M.: An efficient quantum algorithm for colored Jones polynomials arXiv.org:quant-ph/0606167 (2006)

Gascuel, O. McKenzie, A.: Performance Analysis of Hierarchical Clustering Algorithms. J. Classif. **21**, 3–18 (2004)

Gascuel, O.: BIONJ: an Improved Version of the NJ Algorithm Based on a Simple Model of Sequence Data. Mol. Biol. Evol. **14**, 685–695 (1997)

Gascuel, O., Steel, M.: Neighbor-Joining Revealed. Mol. Biol. Evol. **23**, 1997–2000 (2006)

Gąsieniec, L., Jansson, J., Lingas, A.: Efficient approximation algorithms for the hamming center problem. In: Proc. 10th ACM-SIAM Symp. on Discrete Algorithms., pp. 135–S906. (1999)

Gąsieniec, L., Karpinski, M., Plandowski, W., Rytter, W.: Efficient algorithms for Lempel–Ziv encoding. In: Proc. 5th Scandinavian Workshop on Algorithm Theory (SWAT'96). LNCS, vol. 1097, pp. 392–403 (1996)

Gasieniec, L., Kranakis, E., Krizanc, D., Pelc, A.: Minimizing Congestion of Layouts for ATM Networks with Faulty Links. In: Penczek, W., Szalas, A. (eds.) Proceedings of the 21st International Symposium on Mathematical Foundations of Computer Science. Lecture Notes in Computer Science, vol. 1113, pp. 372–381. Springer, Berlin (1996)

Gąsieniec, L., Pelc, A., Peleg, D.: The Wakeup Problem in Synchronous Broadcast Systems (Extended Abstract). In: Proc. of the 19th ACM Symposium on Principles of Distributed Computing (PODC), pp. 113–121 (2000)

Gasieniec, L., Su, C., Wong, P.W.H., Xin, Q.: Routing via single-source and multiple-source queries in static sensor networks. J. Discret. Algorithm **5**(1), 1–11 (2007). A preliminary version of the paper appeared in IPDPS'2005

Geary, R., Raman, R., Raman, V.: Succinct ordinal trees with level-ancestor queries. In: Proc. 15th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 1–10. New Orleans, USA (2004)

Geary, R.F., Rahman, N., Raman, R., Raman, V.: A simple optimal representation for balanced parentheses. Theor. Comput. Sci. **368**(3), 231–246 (2006)

Geffert, V.: Translation of binary regular expressions into nondeterministic ε-free automata with $\mathcal{O}(n \log n)$ transitions. J. Comput. Syst. Sci. **66**(3), 451–472 (2003)

Geiger, L.C.D., Gupta, A., Vlontzos, J.: Dynamic programming for detecting, tracking and matching elastic contours. IEEE Trans. On Pattern Analysis and Machine Intelligence (1995)

Gelfand, Y., Rodriguez, A., Benson, G.: TRDB – The Tandem Repeats Database. Nucl. Acids Res. **35**(suppl. 1), D80–D87 (2007)

Gemmell, P., Lipton, R., Rubinfeld, R., Sudan, M., Wigderson, A.: Self-testing/correcting for polynomials and for approximate functions. In: Proceedings of the Twenty-Third Annual ACM Symposium on Theory of Computing, pp. 32–42. ACM, New York (1991)

Gentile, C.: The robustness of the p-norm algorithms. Mach. Learn. **53**(3) (2002)

Gentile, C., Warmuth, M.K.: Linear hinge loss and average margin. In: Kearns, M.J., Solla, S.A., Cohn, D.A. (eds.) Advances in neural information processing systems 11, p. 225–231. MIT Press, Cambridge (1999)

Georgakopoulos, G.F.: How to splay for log log n-competitiveness. In: Proc. 4th Int'l Workshop on Experimental and Efficient Algorithms (WEA), pp. 570–579 (2005)

George, A., Liu, J.W.H.: Computer solution of large sparse positive definite systems. Prentice-Hall Series in Computational Mathematics, Prentice-Hall Inc. Englewood Cliffs (1981)

Gerards, B., Marchetti-Spaccamela, A. (eds.): Proceedings of the 3rd Workshop on Algorithmic Methods and Models for Optimization of Railways (ATMOS'03) 2003. Electronic Notes in Theoretical Computer Science, vol. 92. Elsevier (2004)

Gershenfeld, N.A., Chuang, I.L.: Bulk spin-resonance quantum computation. Science **275**, 350–356 (1997)

Gfeller, B., Vicari, E.: A Randomized Distributed Algorithm for the Maximal Independent Set Problem in Growth-Bounded Graphs. In: PODC 2007

Ghao, J., Zhang, L.: Well-Separated Pair Decomposition for the Unit Disk Graph Metric and its Applications. SIAM J. Comput. **35**(1), 151–169 (2005)

Giammarresi D., Italiano G.F.: Decremental 2- and 3-connectivity on planar graphs. Algorithmica **16**(3), 263–287 (1996)

Giancarlo, R.: A generalization of the suffix tree to square matrices, with application. SIAM J. Comput. **24**, 520–562 (1995)

Giancarlo, R.: An index data structure for matrices, with applications to fast two-dimensional pattern matching. In: Proceedings of Workshop on Algorithm and Data Structures, vol. 709, pp. 337–348. Springer Lect. Notes Comp. Sci. Montréal, Canada (1993)

Giancarlo, R., Grossi, R.: On the construction of classes of suffix trees for square matrices: Algorithms and applications. Inf. Comput. **130**, 151–182 (1996)

Giancarlo, R., Grossi, R.: Suffix tree data structures for matrices. In: Apostolico, A., Galil, Z. (eds.) Pattern Matching Algorithms, ch. 11„ pp. 293–340. Oxford University Press, Oxford (1997)

Giancarlo, R., Guaiana, D.: On-line construction of two-dimensional suffix trees. J. Complex. **15**, 72–127 (1999)

Giancarlo, R., Restivo, A., Sciortino, M.: From first principles to the Burrows and Wheeler transform and beyond, via combinatorial optimization. Theor. Comput. Sci. **387**(3):236-248 (2007)

Giegerich, R., Kurtz, S., Stoye, J.: Efficient implementation of lazy suffix trees. Softw. Pract. Exp. **33**, 1035–1049 (2003)

Giesen, J.: Curve reconstruction, the TSP, and Menger's theorem on length. Discret. Comput. Geom. **24**, 577–603 (2000)

Gifford, D.K.: Weighted voting for replicated data. In: Proceedings of the 7th ACM Symposium on Operating Systems Principles, 1979, pp. 150–162

Gilbert, A.C., Guha, S., Indyk, P., Muthukrishnan, S., Strauss, M.: Near-optimal sparse fourier representations via sampling. In: Proceedings of the thiry-fourth annual ACM symposium on Theory of computing, pp. 152–161. ACM Press (2002)

Gilbert, A.C., Muthukrishnan, S., Strauss, M.J.: Improved time bounds for near-optimal sparse fourier representation via sampling. In: Proceedings of SPIE Wavelets XI, San Diego, CA 2005 (2005)

Gilbert, A.C., Strauss, M.J., Tropp, J.A., Vershynin, R.: One sketch for all: Fast algorithms for compressed sensing. In: 39th ACM Symposium on Theory of Computing (STOC'07)

Gilbert, E.N., Pollak, H.O.: Steiner minimal trees. SIAM J. Appl. Math. **16**, 1–29 (1968)

Gilbert, S., Lynch, N., Shvartsman, A.: Rambo II: rapidly reconfigurable atomic memory for dynamic networks. In: Proc. International Conference on Dependable Systems and Networks, pp. 259–268. San Francisco, 22–25 June 2003

Gilboa, I., Zemel, E.: Nash and correlated equilibria: some complexity considerations. Games Econ. Behav. **1**, 80–93 (1989)

Gimpel, J.F.: A reduction technique for prime implicant tables. IEEE Trans. Electron. Comput. **14**(4), 535–541 (1965)

Giridhar, A., Kumar, P.R.: Maximizing the Functional Lifetime of Sensor Networks. In: Proceedings of The Fourth International Conference on Information Processing in Sensor Networks, IPSN '05, UCLA, Los Angeles, April 25–27 2005

Gisin, N., Ribordy, G., Tittel, W., Zbinden, H.: Quantum cryptography. Rev. Mod. Phys. **74**, 145–195 (2002)

Gittins, J.C.: Bandit processes and dynamic allocation indices. J. R. Stat. Soc. Series B, **41**(2), 148–177 (1979)

Gittins, J.C., Jones, D.M.: A dynamic allocation index for the sequential design experiments. In: Gani, J., Sarkadu, K., Vince, I. (eds.) Progress in Statistics. European Meeting of Statisticians I, pp. 241–266. North Holland, Amsterdam (1974)

Gkantsidis, C., Rodriguez, P.: Network coding for large scale content distribution. In: IEEE/INFOCOM, 2005

Glazebrook, K., Niño-Mora, J.: Parallel scheduling of multiclass M/M/m queues: approximate and heavy-traffic optimization of achievable performance. Oper. Res. **49**(4), 609–623 (2001)

Gluick, T.C., Draper, D.E.: Thermodynamics of folding a pseudoknotted mRNA fragment. J. Mol. Biol. **241**, 246–262 (1994)

Glushkov, V.M.: The abstract theory of automata. Russ. Math. Surv. **16**, 1–53 (1961)

Goemans, M., Li, L., Mirrokni, V.S., Thottan, M.: Market sharing games applied to content distribution in ad-hoc networks. In: Proceedings of the 5th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc), pp. 1020–1033 (2004)

Goemans, M., Mirrokni, V.S., Vetta, A.: Sink equilibria and convergence. In: 46th Conference on Foundations of Computer Science (FOCS), pp. 123–131 (2005)

Goemans, M., Queyranne, M., Schulz, A., Skutella, M., Wang, Y.: Single machine scheduling with release dates. SIAM J. Discret. Math. **15**, 165–192 (2002)

Goemans, M.X.: Minimum bounded degree spanning trees. In: Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006), 2006, pp. 273–282

Goemans M.X., Goldberg A.V., Plotkin S.A., Shmoys D.B., Tardos É., Williamson D.P.: Improved Approximation Algorithms for Network Design Problems. In: Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 223–232. (1994)

Goemans, M.X., Skutella, M.: Cooperative Facility Location Games. J. Algorithms **50**, 194–214 (2004)

Goemans, M.X., Williamson, D.P.: A general approximation technique for constrained forest problems. SIAM J. Comput. **24**(2), 296–317 (1995)

Goemans, M.X., Williamson, D.P.: Approximation algorithms for MAX-3-CUT and other problems via complex semidefinite programming. STOC 2001 Special Issue of J. Comput. Syst. Sci. **68**, 442–470 (2004)

Goemans, M.X., Williamson, D.P.: Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. J. ACM **42**(6), 1115–1145 (1995)

Goemans, M.X., Williamson, D.P.: New 3/4-approximation algorithms for the maximum satisfiability problem. SIAM J. Discret. Math. **7**, 656–666 (1994)

Goemans, M.X., Williamson, D.P.: The primal-dual method for approximation algorithms and its application to network design problems. In: Hochbaum, D. (ed.) Approximation Algorithms for $\mathcal{NP}$-Hard Problems, Chapter 4, pp. 144–191. PWS Publishing Company, Boston (1996)

Goerdt, A.: A threshold for unsatisfiability. J. Comput. Syst. Sci. **33**, 469–486 (1996)

Goerdt, A.: Random regular graphs with edge faults: Expansion through cores. Theor. Comput. Sci. **264**(1), 91–125 (2001)

Goerdt, A.: The giant component threshold for random regular graphs with edge faults. In: Proceedings of Mathematical Foundations of Computer Science '97 (MFCS'97), pp. 279–288. (1997)

Gold, E.M.: Language identification in the limit. Inform. Control **10**(5), 447–474 (1967)

Goldberg, A.: Selecting problems for algorithm evaluation. In: Proc. 3rd Workshop on Algorithm Engineering (WAE'99). LNCS, vol. 1668. London, United Kingdom, July 19–21, pp. 1–11 (1999)

Goldberg, A., Kaplan, H., Werneck, R.: Better landmarks within reach. In: 9th DIMACS Implementation Challenge Workshop: Shortest Paths. DIMACS Center, Piscataway, NJ, 13–14 Nov 2006

Goldberg, A.V. Hartline, J.D.: Envy-free auctions for digital goods. In: Proceedings of the 4th ACM Conference on Electronic Commerce (EC-03), New York, 9–12 June 2003, pp. 29–35. ACM Press, New York (2003)

Goldberg, A.V.: AVG Lab. http://www.avglab.com/andrew/

Goldberg, A.V.: Scaling algorithms for the shortest path problem. SIAM J. Comput. **21**, 140–150 (1992)

Goldberg, A.V.: Scaling algorithms for the shortest paths problem. SIAM J. Comput. **24**(3), 494–504 (1995)

Goldberg, A.V.: Shortest path algorithms: Engineering aspects. In: Proc. 12th Int'l Symp. on Algorithms and Computation (ISAAC). LNCS, vol. 2223, pp. 502–513. Springer, Berlin (2001)

Goldberg, A.V., Grigoriadis, M.D., Tarjan, R.E.: Use of Dynamic Trees in a Network Simplex Algorithm for the Maximum Flow Problem. Math. Program. **50**(3), 277–290 (1991)

Goldberg, A.V., Harrelson, C.: Computing the Shortest Path: A^* Search Meets Graph Theory. In: Proc. 16th ACM-SIAM Symposium on Discrete Algorithms – SODA, pp. 156–165. ACM, New York and SIAM, Philadelphia (2005)

Goldberg, A.V., Hartline, J.D.: Competitive auctions for multiple digital goods. In: auf der Heide, F.M. (ed.) Algorithms – ESA 2001, 9th Annual European Symposium, Aarhus, Denmark, 28–31 Aug 2001. Lecture Notes in Computer Science, vol. 2161, pp. 416–427. Springer, Berlin (2001)

Goldberg, A.V., Hartline, J.D., Karlin, A.R., Wright, A.: Competitive auctions. Games Econ. Behav. **55**(2), 242–269 (2006)

Goldberg, A.V., Hartline, J.D., Wright, A.: Competitive Auctions and Digital Goods. In: Proceedings of SODA'01, pp. 735–744. Washington D.C., 7–9 January 2001

Goldberg, A.V., Hartline, J.D., Wright, A.: Competitive auctions and digital goods. In: Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA-01), New York, 7–9 January 2001, pp. 735–744. ACM Press, New York (2001)

Goldberg, A.V., Kaplan, H., Werneck, R.: Reach for A*: efficient point-to-point shortest path algorithms. In: Proc. 8th Workshop on Algorithm Engineering and Experiments (ALENEX), 2006

Goldberg, A.V., Radzik, T.: A Heuristic Improvement of the Bellman—Ford Algorithm. Appl. Math. Lett. **6**(3), pp. 3–6 (1993)

Goldberg, A.V., Rao, S.: Beyond the Flow Decomposition Barrier. J. ACM **45**(5), 783–797 (1998)

Goldberg, A.V., Tarjan, R.E.: Solving minimum cost flow problem by successive approximation. In: Proc. ACM Symposium on the Theory of Computing, pp. 7–18 (1987). Full paper in: Math. Oper. Res. **15**, 430–466 (1990)

Goldberg, A.V., Tsioutsiouliklis, K.: Cut Tree Algorithms: An Experimental Study. J. Algorithms **38**(1), 51–83 (2001)

Goldberg, A.W., Tarjan, R.E.: A New Approach to the Maximum Flow Problem. J. SIAM **35**, 921–940 (1988)

Goldberg, L.A., Paterson, M., Srinivasan, A., Sweedyk, E.: Better approximation guarantees for job-shop scheduling. SIAM J. Discret. Math. **14**, 67–92 (2001)

Goldberg, P.W., Golumbic, M.C., Kaplan, H., Shamir, R.: Four strikes against physical mapping of DNA. J. Comput. Biol. **2**(1), 139–152 (1995)

Goldman, S., Kearns, M., Schapire, R.: Exact identification of read-once formulas using fixed points of amplification functions. SIAM J. Comput. **22**(4), 705–726 (1993)

Goldreich, O.: Foundations of Cryptography, vol. 1-2. Cambridge University Press, UK (2001) (2004)

Goldreich, O.: Foundations of Cryptography, Volumes 1 and 2. Cambridge University Press, Cambridge (2001), (2004)

Goldreich, O.: Foundations of Cryptography: Basic Tools. Cambridge University Press (2001)

Goldreich, O.: The Foundations of Cryptography – Volume 1. Cambridge University Press, Campridge, UK (2001)

Goldreich, O., Levin, L.: A hard-core predicate for all one-way functions. In: Proceedings of the 21st ACM Symposium on Theory of Computing, pp. 25–32 Seattle, 14–17 May 1989

Goldreich, O., Petrank, E.: The Best of Both Worlds: Guaranteeing Termination in Fast Randomized Agreement Protocols. Inf. Process. Lett. **36**(1), 45–49 (1990)

Goldschlager, L.M., Shaw, R.A., Staples, J.: The maximum flow problem is log-space complete for P. Theor. Comput. Sci. **21**, 105–111 (1982)

Goldstein, A.J.: An efficient and constructive algorithm for testing whether a graph can be embedded in the plane. In: Graph and Combinatorics Conf. (1963)

Goldwasser, M.H., Kao, M.-Y., Lu, H.-I.: Linear-time algorithms for computing maximum-density sequence segments with bioinformatics applications. J. Comput. Syst. Sci. **70**, 128–144 (2005)

Golubchik, L., Khanna, S., Khuller, S., Thurimella, R., Zhu, A.: Approximation algorithms for data placement on parallel disks. In: Symposium on Discrete Algorithms, pp. 223–232. Society for Industrial and Applied Mathematics, Philadelphia (2000)

Golubchik, L., Khuller, S., Kim, Y., Shargorodskaya, S., Wan., Y.: Data migration on parallel disks. In: 12th Annual European Symposium on Algorithms (ESA) (2004)

Golumbic, M.C.: Combinatorial Merging. IEEE Trans. Comput. **C-25**, 1164–1167 (1976)

Golynski, A.: Optimal lower bounds for rank and select indexes. In: Proc. ICALP 2006, Part I. LNCS, vol. 4051, pp. 370–381 (2006)

Golynski, A., Munro, J.I., Rao, S.S.: Rank/select operations on large alphabets: a tool for text indexing. In: Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 368–373. ACM, SIAM (2006)

Gomory, R.E., Hu, T.C.: Multi-terminal network flows. J. Soc. Indust. Appl. Math. **9**(4), 551–570 (1961)

Gonnet, G.: Expected length of the longest probe sequence in hash code searching. J. Assoc. Comput. Mach. **28**(2), 289–304 (1981)

Gonnet, G., Baeza-Yates, R., Snider, T.: New indices for text: PAT trees and PAT arrays. In: Frakes, W.B., Baeza-Yates, R. (eds.) Information Retrieval: Data Structures & Algorithms. pp. 66–82 Prentice-Hall, Englewood Cliffs (1992)

Gonnet, G.H.: Efficient searching of text and pictures. Tech. Report OED-88-02, University of Waterloo (1988)

González, R., Navarro, G.: Statistical encoding of succinct data structures. In: Proc. CPM 2006. LNCS, vol. 4009, pp. 294–305. Springer, Berlin (2006)

Gonzalez, T., Zheng, S.Q.: Bounds for partitioning rectilinear polygons. In: Proc. 1st Symp. on Computational Geometry (1985)

Gonzalez, T., Zheng, S.Q.: Improved bounds for rectangular and guillotine partitions. J. Symb. Comput. **7**, 591–610 (1989)

Gonzalez, T.F.: Handbook of Approximation Algorithms and Metaheuristics. Chapman & Hall/CRC Computer & Information Science Series (2007)

Gordon, D.: Discrete logarithms in GF(p) using the number field sieve. SIAM J. Discret. Math. **6**(1), 124–139 (1993)

Gormley, T., Reingold, N., Torng, E., Westbrook, J.: Generating adversaries for request-answer games. In: Proc. of the 11th Symposium on Discrete Algorithms. (SODA2000), pp. 564–565 (2000)

Gotoh, O.: An improved algorithm for matching biological sequences. J. Mol. Biol. **162**, 705–708 (1982)

Gottesman, D.: Class of quantum error-correcting codes saturating the quantum Hamming bound. Phys. Rev. A **54**, 1862–1868 (1996)

Gottesman, D.: Stabilizer codes and quantum error correction, Ph. D. thesis, Caltech. (1997) See also: arXiv preprint quant-ph/9705052

Gottlob, G., Scarcello, F., Sideri, M.: Fixed-parameter complexity in AI and nonmonotonic reasoning. Artif. Intell. **138**, 55–86 (2002)

Gottlob, G., Szeider, S.: Fixed-parameter algorithms for artificial intelligence, constraint satisfaction, and database problems. Comput. J., Special Issue on Parameterized Complexity, Advanced Access (2007)

Gowland, P., Lester, D.: Asurvey of exact arithmetic implementations. In: Blank, J., Brattka, V., Hertling, P. (eds.) Computability and Complexity in Analysis, pp. 30–47. Springer, 4th International Workshop, CCA 2000, Swansea, UK, September 17–19, (2000), Selected Papers. Lecture Notes in Computer Science, No. 2064

Grable, D.A., Panconesi, A.: Fast distributed algorithms for Brooks–Vizing colorings. J. Algorithms **37**, 85–120 (2000)

Graefe, G.: B-tree indexes for high update rates. SIGMOD RECORD **35**, 39–44 (2006)

Graham, R.L.: Bounds for certain multiprocessing anomalies. Bell Syst. Techn. J. **45**, 1563–1581 (1966)

Graham, R.L.: Bounds on multiprocessing timing anomalies. SIAM J. Appl. Math. **17**, 263–269 (1969)

Graham, R.L., Hell, P.: On the history of the minimum spanning tree problem. Ann. Hist. Comput. **7**(1), 43–57 (1985)

Graham, R.L., Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G.: Optimization and approximation in deterministic sequencing and scheduling: a survey. Ann. Discret. Math. **5**, 287–326 (1979)

Gramm, J., Guo, J., Hüffner, F., Niedermeier, R.: Automated generation of search tree algorithms for hard graph modification problems. Algorithmica **39**, 321–347 (2004)

Gramm, J., Guo, J., Hüffner, F., Niedermeier, R.: Graph-modeled data clustering: Exact algorithms for clique generation. Theor. Comput. Syst. **38**, 373–392 (2005)

Gramm, J., Niedermeier, R.: Faster exact solutions for Max2Sat. In: Proceedings of CIAC. LNCS, vol. 1767, pp. 174–186. Springer, Berlin (2000)

Grandoni, F.: Exact Algorithms for Hard Graph Problems. Ph. D. thesis, Università di Roma "Tor Vergata", Roma, Italy (2004)

Granot, D., Granot, F., Zhu, W.R.: Characterization Sets for the Nucleolus. Int. J. Game Theory **27**, 359–374 (1998)

Grantson, M.: Fixed-parameter algorithms and other results for optimal partitions. Lecentiate Thesis, Department of Computer Science, Lund University (2004)

Grantson, M., Borgelt, C., Levcopoulos, C.: A fixed parameter algorithm for minimum weight triangulation: Analysis and experiments. Tech. Rep. 154, Department of Computer Science, Lund University (2005)

Grantson, M., Borgelt, C., Levcopoulos, C.: Fixed parameter algorithms for the minimum weight triangulation problem. Tech. Rep. 158, Department of Computer Science, Lund University (2006)

Grantson, M., Borgelt, C., Levcopoulos, C.: Minimum weight triangulation by cutting out triangles. In: Deng, X., Du, D.-Z. (eds.) Proceedings of the 16th Annual International Symposium on Algorithms and Computation (ISAAC). Lecture Notes in Computer Science, vol. 3827, pp. 984–994. Springer, New York (2005)

Grantson, M., Borgelt, C., Levcopoulos, C.: Minimum Weight Triangulation by Cutting Out Triangles. In: Proceedings 16th Annual International Symposium on Algorithms and Computation, ISAAC 2005, Sanya, China, pp. 984–994. Lecture Notes in Computer Science, vol. 3827. Springer, Heidelberg (2005)

Grantson, M., Levcopoulos, C.: A fixed parameter algorithm for the minimum number convex partition problem. In: Akiyama, J., Kano, M., Tan, X. (eds.) Proceedings of Japanese Conference on Discrete and Computational Geometry (JCDCG 2004). Lecture Notes in Computer Science, vol. 3742, pp. 83–94. Springer, New York (2005)

Graunke, G., Thakkar, S.: Synchronization algorithms for shared-memory multiprocessors. IEEE Comput. **28**(6), 69–69 (1990)

Gray, J.: A Comparison of the Byzantine Agreement Problem and the Transaction Commit Problem. In: Fault-Tolerant Distributed Computing [Asilomar Workshop 1986]. LNCS, vol. 448, pp. 10–17. Springer, Berlin (1990)

Grebinski, V., Kucherov, G.: Optimal Query Bounds for Reconstructing a Hamiltonian Cycle in Complete Graphs. Proc. 5th Israeli Symposium on Theoretical Computer Science, pp. 166–173. (1997)

Grebinski, V., Kucherov, G.: Reconstructing a Hamiltonian Cycle by Querying the Graph: Application to DNA Physical Mapping. Discret. Appl. Math. **88**, 147–165 (1998)

Greene, D.H., Yao, F.F.: Finite-resolution computational geometry. IEEE Found. Comput. Sci. **27**, 143–152 (1986)

Griffith, J., Robins, G., Salowe, J.S., Zhang, T.: Closing the gap: Near-optimal steiner trees in polynomial time. IEEE Transac. Comput. Aided Des. **13**, 1351–1365 (1994)

Griggs, J., Liu, D.: Minimum Span Channel Assignments. In: Recent Advances in Radio Channel Assignments, Invited Minisymposium, Discrete Mathematics (1998)

Grigoriadis, M.D., Khachiyan, L.G.: Coordination complexity of parallel price-directive decomposition. Mathematics of Operations Research, **21**, 321–340. (1996)

Grigoriadis, M.D., Khachiyan, L.G.: Fast approximation schemes for convex programs with many blocks and coupling constraints. SIAM J. Optim. **4**, 86–107 (1994)

Grigoriev, A., Sviridenko, M., Uetz, M.: Machine scheduling with resource dependent processing times. Math. Program. **110**(1B), 209–228 (2002)

Grigoriev, D.: Testing Shift-Equivalence of Polynomials by Deterministic, Probabilistic and Quantum Machines. Theor. Comput. Sci. **180**, 217–228 (1997)

Grishman, R., Yangarber, R.: Private Communication. NYU (1995)

Grodstein, J., Lehman, E., Harkness, H., Grundmann, B., Watanabe, Y.: A delay model for logic synthesis of continuously sized networks. In: Proceedings of the 1995 International Conference on Computer-Aided Design, pp. 458–462. November 1995

Grohe, M.: Local tree-width, excluded minors, and approximation algorithms. Combinatorica **23**(4), 613–632 (2003)

Gromov, M.: Structures Métriques des Variétés Riemanniennes. Textes Math. CEDIX, vol. 1. F. Nathan, Paris (1981)

Gropp, W., Lusk, E., Doss, N., Skjellum, A.: A high-performance, portable implementation of the MPI message passing interface standard. Technical report, Argonne National Laboratory, Argonne, IL, (1996) www.mcs.anl.gov/mpi/mpich/

Grossglauser, M., Tse, D.N.C.: Mobility increases the capacity of ad hoc wireless networks. IEEE/ACM Trans. Netw. **10**(4), 477–486 (2002)

Grossi, R., Gupta, A., Vitter, J.: High-order entropy-compressed text indexes. In: Proc. 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), Baltimore, 12–14 January, pp. 841–850 (2003)

Grossi, R., Gupta, A., Vitter J.S.: High-order entropy-compressed text indexes. In: Farach-Colton, M. (ed) Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, pp. 841–850, Philadelphia (2003)

Grossi, R., Italiano, G.F.: Efficient techniques for maintaining multidimensional keys in linked data structures. In: Proceedings of the 26th International Colloquium on Automata, Languages and Programming (ICALP '99). LNCS, vol. 1644, pp. 372–381. Springer, Prague (1999)

Grossi, R., Vitter, J.: Compressed suffix arrays and suffix trees with applications to text indexing and string matching. SIAM J. Comput. **35**(2), 378–407 (2006)

Grossi, R., Vitter, J.S.: Compressed suffix arrays and suffix trees with applications to text indexing and string matching. In: Proceedings of Symposium on Theory of Computing, 2000, pp. 397–406

Grotschel, M., Lovász, L., Schrijver, A.: Geometric algorithms and combinatorial optimization. Algorithms and Combinatorics, vol. 2, 2nd edn. Springer (1993)

Grötschel, M., Lovász, L., Schrijver, A.: The ellipsoid method and its consequences in combinatorial optimization. Combinatorica **1**, 169–197 (1981)

Grötschel, M., Monma, C.L., Stoer, M.: Computational results with a cutting plane algorithm for designing communication networks with low-connectivity constraints. Oper. Res. **40**(2), 309–330 (1992)

Grötschel, M., Monma, C.L., Stoer, M.: Design of survivable networks. In: Handbooks in Operations Research and Management Science, vol. 7, Network Models, chapter 10, pp. 617–672. North-Holland, Amsterdam (1995)

Grover, L.K.: A fast quantum mechanical algorithm for database search. In: Proceedings of the 28th ACM Symposium on the Theory of Computing, pp. 212–219, Philadelphia, PA, USA, 22–24 May 1996

Grover, L.K.: A framework for fast quantum mechanical algorithms. In: Proc. 30th ACM Symp. on Theory of Computing (STOC), pp. 53–62. Dallas, 23–26 May 1998

Grover, L.K.: Fixed-point quantum search. Phys. Rev. Lett. **95**, 150501 (2005)

Grüne, A.: Geometric Dilation and Halving Distance. Ph. D. thesis, Institut für Informatik I, Universität Bonn (2006)

Gu, Q.P., Peng, S., Sudborough, H.: A 2-approximation algorithm for genome rearrangements by reversals and transpositions. Theor. Comput. Sci. **210**, 327–339 (1999)

Gu, Q.P., Tamaki, H.: Branch-width, parse trees, and monadic second-order logic for matroids. J. Combin. Theor. Ser. B **96**, 325–351 (2006)

Gu, Q.P., Tamaki, H.: Optimal branch-decomposition of planar graphs in O(n^3) time. In: Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP 2005). Lecture Notes Computer Science, vol. 3580, pp. 373–384. Springer, Berlin (2005)

Guan, X.Y.: Face traversal routing on edge dynamic graphs. In: Proceedings of the Nineteenth International Parallel and Distributed Processing Symposium, Denver, Colorado, April 2005

Gubbala, P., Raghavachari, B.: A 4/3-approximation algorithm for minimum 3-edge-connectivity. In: Proceedings of the Workshop on Algoriths and Data Structures (WADS) August 2007, pp. 39–51. Halifax (2007)

Gubbala, P., Raghavachari, B.: Approximation algorithms for the minimum cardinality two-connected spanning subgraph problem. In: Jünger, M., Kaibel, V. (eds.) IPCO. Lecture Notes in Computer Science, vol. 3509, pp. 422–436. Springer, Berlin (2005)

Gudmundsson, J., Levcopoulos, C.: A Parallel Approximation Algorithm for Minimum Weight Triangulation. Nordic J. Comput. **7**(1), 32–57 (2000)

Gudmundsson, J., Levcopoulos, C., Narasimhan, G.: Fast greedy algorithms for constructing sparse geometric spanners. SIAM J. Comput. **31**, 1479–1500 (2002)

Gudmundsson, J., Levcopoulos, C., Narasimhan, G., Smid, M.: Approximate distance oracles for geometric graphs. In: Proceedings of the 13th ACM-SIAM Symposium on Discrete Algorithms, pp. 828–837. ACM Press, New York (2002)

Gudmundsson, J., Levcopoulos, C., Narasimhan, G., Smid, M.: Approximate distance oracles for geometric spanners, ACM Trans. Algorithms (2008). To Appear

Gudmundsson, J., Levcopoulos, C., Narasimhan, G., Smid, M.: Approximate distance oracles revisited. In: Proceedings of the 13th International Symposium on Algorithms and Computation. Lecture Notes in Computer Science, vol. 2518, Berlin, pp. 357–368. Springer, London (2002)

Gudmundsson, J., Narasimhan, G., Smid, M.: Fast pruning of geometric spanners. In: Proceedings of the 22nd Symposium on Theoretical Aspects of Computer Science. Lecture Notes in Computer Science, vol. 3404, Berlin, pp. 508–520. Springer, London (2005)

Guerraoui, R.: Indulgent algorithms. In: Proceedings of the 19th Annual ACM Symposium on Principles of Distributed Computing, Portland, Oregon, USA, pp. 289–297, ACM, July 2000

Guerraoui, R., Herlihy, M., Pochon, B.: A topological treatment of early-deciding set-agreement. In: OPODIS, pp. 20–35, (2006)

Guerraoui, R., Herlihy, M., Pochon, B.: Polymorphic contention management. In: Proc. 19th Annual International Symposium on Distributed Computing, 2005

Guerraoui, R., Herlihy, M., Pochon, B.: Toward a theory of transactional contention managers. In: Proc. 24th Annual ACM Symposium on Principles of Distributed Computing, 2005, pp. 258–264

Guerraoui, R., Kapalka, M., Kouznetsov, P.: The weakest failure detector to boost obstruction freedom. In: Proc. 20th Annual International Symposium on Distributed Computing, 2006

Guerraoui, R., Ruppert, E.: Anonymous and fault-tolerant shared-memory computing. Distrib. Comput. **20**(3) 165–177 (2007)

Guestrin, C., Koller, D., Parr, R., Venkataraman, S.: Efficient solution algorithms for factored mdps. J. Artif. Intell. Res. **19**, 399–468 (2003)

Guha, S., Khuller, S.: Approximation algorithms for connected dominating sets. Algorithmica **20**, 374–387 (1998)

Guha, S., Khuller, S.: Greedy strikes back: Improved facility location algorithms. In: Proceedings of the 9th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 228–248. SIAM, Philadelphia (1998)

Guha, S., Khuller, S.: Greedy strikes back: Improved facility location algorithms. J. Algorithms **31**, 228–248 (1999)

Guha, S., Meyerson, A., Munagala, K.: A constant factor approximation for the single sink edge installation problem. In: Proceedings of the 33rd Annual ACM Symposium on Theory of Computing (STOC), pp. 383–388. ACM Press, New York (2001)

Guha, S., Meyerson, A., Munagala, K.: Hierarchical placement and network design problems. In: Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp. 603–612. IEEE Computer Society, Los Alamitos, CA, USA (2000)

Guhaa, S., Khuller, S.: Improved methods for approximating node weighted Steiner trees and connected dominating sets. Inf. Comput. **150**, 57–74 (1999)

Gui, H., Muller, R., Vohra, R.V.: Characterizing dominant strategy mechanisms with multi-dimensional types (2004). Working paper

Guibas, L.: Kinetic data structures: A state of the art report. In: Proc. 3rd Workshop on Algorithmic Foundations of Robotics, pp. 191–209 (1998)

Guibas, L.: Modeling Motion. In: Goodman, J., O'Rourke, J.: (eds), Handbook of Discrete and Computational Geometry. CRC Press, 2nd ed. (2004)

Guibas, L., Ngyuen, A., Russel, D., Zhang, L.: Collision detection for deforming necklaces. In: Proc. 18th ACM Symposium on Computational Geometry, 2002, pp. 33–42

Guibas, L., Salesin, D., Stolfi, J.: Epsilon geometry: building robust algorithms from imprecise computations. ACM Symp Comput. Geometr. **5**, 208–217 (1989)

Guillemot, S., Berry, V.: Fixed-parameter tractability of the maximum agreement supertrees. In: Proceedings of the 18th Annual Symposium on Combinatorial Pattern Matching (CPM 2007). Lecture Notes in Computer Science. Springer, (2007)

Guillemot, S., Nicolas, F.: Solving the maximum agreement subtree and the maximum compatible tree problems on many bounded degree trees. In: Lewenshtein, M., Valiente, G. (eds.) Proc. of the 17th Combinatorial Pattern Matching Symposium (CPM'06). LNCS, vol. 4009, pp. 165–176. Springer, Berlin (2006)

Gummadi, K., Gummadi, R., Gribble, S., Ratnasamy, S., Shenker, S., Stoica, I.: The impact of DHT routing geometry on resilience and proximity. In: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications, pp. 381–394. ACM Press (2003)

Gummadi, K.P., Dunn, R.J., Saroiu, S., Gribble, S.D., Levy, H.M., Zahorjan, J.: Measurement, modeling, and analysis of a peer-topeer file-sharing workload. In: Proceedings of the nineteenth ACM symposium on Operating systems principles, pp. 314–329. ACM Press (2003)

Gunopolous, D., Khardon, R., Mannila, H., Saluja, S., Toivonen, H., Sharma, R.S.: Discovering All Most Specific Sentences. ACM Trans. Database Syst. **28**, 140–174 (2003)

Guo, J., Gramm, J., Hüffner, F., Niedermeier, R., Wernicke, S.: Compression-based fixed-parameter algorithms for Feedback Vertex Set and Edge Bipartization. J. Comp. Syst. Sci. **72**(8), 1386–1396 (2006)

Guo, J., Niedermeier, R.: Invitation to data reduction and problem kernelization. ACM SIGACT News **38**(1), 31–45 (2007)

Guo, J., Niedermeier, R.: Linear problem kernels for NP-hard problems on planar graphs. In: Proc. 34th ICALP. LNCS, vol. 4596, pp. 375–386. Springer, Berlin (2007)

Guo, J., Niedermeier, R., Wernicke, S.: Fixed-parameter tractability results for full-degree spanning tree and its dual. In: Proc. 2nd IWPEC. LNCS, vol. 4196, pp. 203–214. Springer, Berlin (2006)

Guo, P.N., Cheng, C.K., Yoshimura, T.: An O-tree representation of non-slicing floorplan and its applications. In: 36th DAC., June 1998, pp. 268–273

Guo, W., Liu, Z., Wu, G.: An Energy-Balanced Transmission Scheme for Sensor Networks. In: 1st ACM International Conference on Embedded Networked Sensor Systems (ACM SenSys 2003), Poster Session, Los Angeles, CA, November 2003

Gupta, A.: Formal Hardware Verification Methods: A Survey. Formal Method Syst. Des. **1**, 151–238 (1993)

Gupta, A.: Improved bandwidth approximation for trees and chordal graphs. J. Algorithms **40**(1), 24–36 (2001)

Gupta, A.: Steiner points in tree metrics don't (really) help. In: SODA '01: Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms, Philadelphia, PA, USA, Society for Industrial and Applied Mathematics, pp. 220–227. (2001)

Gupta, A., Hajiaghayi, M.T., Räcke, H.: Oblivious network design. In: SODA '06: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm, pp. 970–979. ACM Press, New York (2006)

Gupta, A., Hon, W.K., Shah, R., Vitter, J.S.: Compressed data structures: Dictionaries and data-aware measures. In: Storer, J.A., Cohn, M. (eds) Proc. 16th IEEE Data Compression Conference, pp. 213–222, IEEE, Snowbird, Utah, March 2006 Computer Society, Los Alamitos, CA

Gupta, A., Kumar, A., Pál, M., Roughgarden, T.: Approximation via cost-sharing: a simple approximation algorithm for the multicommodity rent-or-buy problem. In: Proc. of the 44th Annual IEEE Symposium on Foundations of Computer Science, pp. 606–617., IEEE Computer Society, Washington (2003)

Gupta, A., Kumar, A., Pál, M., Roughgarden, T.: Approximation via cost-sharing: simpler and better approximation algorithms for network design. J. ACM 54(3), Article 11 (2007)

Gupta, A., Pál, M., Ravi, R., Sinha, A.: Boosted sampling: approximation algorithms for stochastic optimization. In: Proceedings of the 36st Annual ACM Symposium on Theory of Computing (STOC), pp. 417–426. ACM, New York (2004)

Gupta, A., Talwar, K.: Approximating unique games. In: SODA '06: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm, New York, NY, USA, pp. 99–106. ACM Press, New York (2006)

Gupta, P., Kumar, P.R.: The Capacity of Wireless Networks. IEEE Trans. Inf. Theory, IT-**46**(2), 388–404 (2000)

Guruswami, V.: Algorithmic Results in List Decoding. In: Foundations and Trends in Theoretical Computer Science, vol. 2, issue 2, NOW publishers, Hanover (2007)

Guruswami, V.: List Decoding of Error-Correcting Codes. Lecture Notes in Computer Science, vol. 3282. Springer, Berlin (2004)

Guruswami, V., Indyk, P.: Linear-time encodable/decodable codes with near-optimal rate. IEEE Trans. Inf. Theory **51**(10), 3393–3400 (2005)

Guruswami, V., Khanna, S.: On the hardness of 4-coloring a 3-colorable graph. In: Proceedings of the 15th annual IEEE Conference on Computational Complexity (2000) pp. 188–197.

Guruswami, V., Khanna, S., Rajaraman, R., Shepherd, F.B., Yannakakis, M.: Near-Optimal Hardness Results and Approximation Algorithms for Edge-Disjoint Paths and Related Problems. J. CSS **67**, 473–496 (2003). Preliminary version in Proc. of ACM STOC 1999

Guruswami, V., Patthak, A.: Correlated Algebraic-Geometric codes: Improved list decoding over bounded alphabets. In: Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp. 227–236, Berkley, October 2006

Guruswami, V., Raghavendra, P.: Hardness of Learning Halfspaces with Noise. In: Proceedings of FOCS, pp. 543–552 (2006)

Guruswami, V., Rudra, A.: Explicit capacity-achieving list-decodable codes. In: Proceedings of the 38th Annual ACM Symposium on Theory of Computing, pp. 1–10. Seattle, May 2006

Guruswami, V., Rudra, A.: Explicit codes achieving list decoding capacity: Error-correction with optimal redundancy. IEEE Trans. Inform. Theor. **54**(1), 135–150 (2008)

Guruswami, V., Rudra, A.: Limits to list decoding Reed–Solomon codes. IEEE Trans. Inf. Theory. **52**(8), 3642–3649 (2006)

Guruswami, V., Sudan, M.: Improved decoding of Reed–Solomon and algebraic-geometric codes. IEEE Trans. Inf. Theory. **45**(6), 1757–1767 (1999)

Guruswami, V., Vardy A.: Maximum Likelihood Decoding of Reed–Solomon codes is NP-hard. IEEE Trans. Inf. Theory. **51**(7), 2249–2256 (2005)

Gusfield, D. Orzack, S.H.: Haplotype inference. In: Aluru S. (ed) Handbook of Computational Molecular Biology, pp. 1–28. Champman and Hall/CRC-press, Boca Raton (2005)

Gusfield, D.: Algorithms on Strings, Trees and Sequences. Cambridge University Press, Cambridge (1997)

Gusfield, D.: Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology. Cambridge University Press, New York (1997)

Gusfield, D.: Efficient methods for multiple sequence alignment with guaranteed error bounds. Bull. Math. Biol. **55**(1), 141–154 (1993)

Gusfield, D.: Haplotyping as perfect phylogeny: Conceptual framework and efficient solutions. In: Myers, G., Hannenhalli, S., Istrail, S., Pevzner, P., Waterman, M. (eds.) Proceedings of the Annual International Conference on Computational Molecular Biology (RECOMB), New York, 2002, pp. 166–175. ACM Press (2002)

Gusfield, D.: The structure of the stable roommate problem: efficient representation and enumeration of all stable assignments. SIAM J. Comput. **17**(4), 742–769 (1988)

Gusfield, D.: Three fast algorithms for four problems in stable marriage. SIAM J. Comput. **16**(1), 111–128 (1987)

Gusfield, D.: Very Simple Methods for All Pairs Network Flow Analysis. SIAM J. Comput. **19**(1), 143–155 (1990)

Gusfield, D., Eddhu, S., Langley, C.: Efficient reconstruction of phylogenetic networks with constrained recombination. In: Proc. of Computational Systems Bioinformatics (CSB2003), 2003 pp. 363–374

Gusfield, D., Irving, R.W.: The Stable Marriage Problem. Structure and Algorithms. MIT Press, Cambridge (1989)

Gusfield, D.and Stoye, J.: Linear time algorithms for finding and representing all the tandem repeats in a string. J. Comput. Syst. Sci. **69**(4), 525–546 (2004)

Gusfield, D.M.: Efficient algorithms for inferring evolutionary trees. Networks **21**, 19–28 (1991)

Gutman, R.: Reach-based Routing: A New Approach to Shortest Path Algorithms Optimized for Road Networks. In: Algorithm Engineering and Experiments – ALENEX (SIAM, 2004), pp. 100–111. SIAM, Philadelphia (2004)

Guttman, A.: R-trees: A dynamic index structure for spatial searching. In: Proc. SIGMOD International Conference on Management of Data, 1984, pp. 47–57

Gyárfás, A., Lehel, J.: Effective on-line coloring of P_5-free graphs. Combinatorica **11**(2), 181–184 (1991)

Hadzilacos, V., Toueg, S.: Fault-tolerant broadcasts and related problems. In: Mullender, S. (ed.) Distributed Systems, 2nd edn., pp. 97–146. ACM Press Books, Addison-Wesley (1993). Extended version appeared as Cornell Univ. TR 94-1425

Hagen, L., Kahng, A.B.: Fast Spectral Methods for Ratio Cut Partitioning and Clustering. In: Proc. IEEE Int. Conf. on Computer-Aided Design, November 1991, pp. 10–13

Hagerup, T.: Improved shortest paths on the word RAM. In: Proc. 27th Int'l Colloq. on Automata, Languages, and Programming (ICALP). LNCS vol. 1853, pp. 61–72. Springer, Berlin (2000)

Hahne, E.: Round Robin Scheduling for Max-min Fairness in Data Networks. IEEE J. Sel. Areas Commun. **9**(7), 1024–1039 (1991)

Hajiaghayi, M., Kleinberg, R., Parkes, D.: Adaptive limited-supply online auctions. In: Proc. of the 6th ACM Conference on Electronic Commerce (EC'04), 2004

Hajiaghayi, M., Mahdian, M., Mirrokni, V.S.: The facility location problem with general cost functions. Netw. **42**(1), 42–47 (2003)

Hajiaghayi, M.T., Kim, J.H., Leighton, T., Räcke, H.: Oblivious routing in directed graphs with random demands. In: Proceedings of the 37th Annual ACM Symposium on Theory of Computing, 2005, pp. 193–201

Hajiaghayi, M.T., Kleinberg, R.D., Leighton, T., Räcke, H.: Oblivious routing on node-capacitated and directed graphs. In: Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms, 2005, pp. 782–790

Hakimi, S.L., Yau, S.S.: Distance matrix of a graph and its realizability. Quarterly Appl. Math. **22**, 305–317 (1964)

Haldar, S., Vidyasankar, K.: Constructing 1-writer multireader multivalued atomic variables from regular variables. J. ACM **42**(1), 186–203 (1995)

Haldar, S., Vitanyi, P.: Bounded concurrent timestamp systems using vector clocks. J. Assoc. Comp. Mach. **49**(1), 101–126 (2002)

Hale, W. K.: Frequency assignment: Theory and applications. Proc. IEEE. **68**(12), 1497–1513 (1980)

Hale, W.K.: Frequency Assignment: Theory and Applications. In: Proceedings of the IEEE, vol. 68, number 12, pp. 1497-1514 (1980)

Hales, L., Hallgren, S.: An improved quantum Fourier transform algorithm and applications. In: Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science, pp. 515–525 (2000)

Hall, J., Hartline, J., Karlin, A., Saia, J., Wilkes, J.: On algorithms for efficient data migration. In: SODA, pp. 620–629. Society for Industrial and Applied Mathematics, Philadelphia (2001)

Hall, L.A., Schulz, A.S., Shmoys, D.B., Wein, J.: Scheduling to minimize average completion time: off-line and on-line approximation algorithms. Math. Oper. Res. **22**(3), 513–544 (1997)

Halldorsson, B.V., Bafna, V., Edwards, N., Lippert, R., Yooseph, S., Istrail, S.: A survey of computational methods for determining haplotypes. In: Computational methods for SNP and haplotype inference: DIMACS/RECOMB satellite workshop. Lecture Notes in Computer Science, vol. 2983, pp. 26–47. Springer, Berlin (2004)

Halldorsson, M.: A still better performance guarantee for approximate graph coloring. Inf. Process. Lett. **45**, 19–23 (1993)

Halldorsson, M., Karlsson, R.: Strip graphs: Recognition and scheduling. In: Proc. 32nd International Workshop on Graph-Theoretic Concepts in Computer Science (WG). Lecture Notes in Computer Science, vol. 4271, pp. 137–146. Springer, Berlin (2006)

Halldórsson, M.M., Irving, R.W., Iwama, K., Manlove, D.F., Miyazaki, S., Morita, Y., Scott, S.: Approximability results for stable marriage problems with ties. Theor. Comput. Sci. **306**, 431–447 (2003)

Halldórsson, M.M., Iwama, K., Miyazaki, S., Yanagisawa, H.: Improved approximation of the stable marriage problem. Proc. ESA 2003. LNCS 2832, pp. 266–277. (2003)

Halldórsson, M.M., Iwama, K., Miyazaki, S., Yanagisawa, H.: Randomized approximation of the stable marriage problem. Theor. Comput. Sci. **325**(3), 439–465 (2004)

Hallgren, S.: Fast quantum algorithms for computing the unit group and class group of a number field. In: Proceedings of the 37th ACM Symposium on Theory of Computing. (2005)

Hallgren, S.: Polynomial-time quantum algorithms for Pell's equation and the principal ideal problem. J. ACM **54**(1), 1–19 (2007)

Halperin, D., Leiserowitz, E.: Controlled perturbation for arrangements of circles. Int. J. Comput. Geom. Appl. **14**(4–5), 277–310 (2004)

Halperin, E., Nathaniel, R., Zwick, U.: Coloring k-colorable graphs using smaller palettes. J. Algorithms **45**, 72–90 (2002)

Halperin, E., Zwick, U.: A unified framework for obtaining improved approximation algorithms for maximum graph bisection problems. Random Struct. Algorithms **20**(3), 382–402 (2002)

Halperin, S., Zwick, U.: Linear time deterministic algorithm for computing spanners for unweighted graphs. unpublished manuscript (1996)

Halpern, J.Y., Megiddo, N., Munshi, A.A.: Optimal precision in the presence of uncertainty. J. Complex. **1**, 170–196 (1985)

Hamdy, S., Maurer, M.: Feige-fiat-shamir identification based on real quadratic fields, Tech. Report TI-23/99. Technische Universität Darmstadt, Fachbereich Informatik, http://www.informatik.tu-darmstadt.de/TI/Veroeffentlichung/TR/ (1999)

Hamel, A.M., Steel, M.A.: Finding a maximum compatible tree is NP-hard for sequences and trees. Appl. Math. Lett. **9**(2), 55–59 (1996)

Hamming, R.: Error detecting and error correcting codes. Bell Syst. Tech. J. **29**, 147–160 (1950)

Han, Y.: Deterministic sorting in $O(n \log\log n)$ time and linear space. J. Algorithms **50**(1), 96–105 (2004). Announced at STOC'02

Han, Y.: Improved fast integer sorting in linear space. Inf. Comput. **170**(8), 81–94 (2001). Announced at STACS'00 and SODA'01

Han, Y.: Improving the Efficiency of Sorting by Reversals, Proceedings of The 2006 International Conference on Bioinformatics and Computational Biology. Las Vegas, Nevada, USA (2006)

Han, Y., Thorup, M.: Integer sorting in $O(n\sqrt{\log\log n})$ expected time and linear space. In: Proceedings of the 43rd Annual Symposium on Foundations of Computer Science (FOCS '02), pp. 135–144. IEEE Computer Society Press, Vancouver (2002)

Hancart, C.: On Simon's string searching algorithm. Inf. Process. Lett. **47**(2), 95–99 (1993)

Hancock, T., Jiang, T., Li, M., Tromp, J.: Lower bounds on learning decision lists and trees. In: 12th Annual Symposium on Theoretical Aspects of Computer Science, pp. 527–538 (1995)

Hanisch, D., Zimmer, R., Lengauer, T.: ProML – the Protein Markup Language for specification of protein sequences, structures and families. In: Silico Biol. **2**, 0029 (2002). http://www.bioinfo.de/isb/2002/02/0029/

Hankerson, D., Menezes, A., Vanstone, S.: Guide to Elliptic Curve Cryptography. Springer, New York (2004)

Hannenhalli, S., Pevzner, P.: Transforming cabbage into turnip (polynomial algorithm for sorting signed permutations by reversals). J. ACM **46**, 1–27 (1999)

Hannenhalli, S., Pevzner, P.A.: Transforming cabbage into turnip (polynomial algorithm for sorting signed permutations by reversals). In: Proc. 27th Ann. Symp. Theory of Computing (STOC95), pp. 178–189. ACM, Las Vegas, NV (1995)

Hansen, P., Thisse, J.F.: Outcomes of voting and planning: condorcet, weber and rawls locations. J. Publ. Econ. **16**, 1–15 (1981)

Haran, I., Halperin, D.: An experimental study of point location in general planar arrangements. In: Proceedings of 8th Workshop on Algorithm Engineering and Experiments, pp. 16–25 (2006)

Harary, F.: Graph Theory. Addison-Wesley, Reading (1969)

Harary, F., Moser, L.: The theory of round robin tournaments. Am. Math. Mon. **73**(3), 231–246 (1966)

Harel, D., Tarjan, R.E.: Fast algorithms for finding nearest common ancestors. SIAM J. Comput. **13**(2), 338–355 (1984)

Hariharan, R., Kavitha, T., Panigrahi, D.: Efficient Algorithms for Computing All Low *s*-*t* Edge Connectivities and Related Problems. In: Proc. of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms, 2007, pp. 127–136

Harrelson, C., Hildrum, K., Rao, S.: A polynomial-time tree decomposition to minimize congestion. In: Proceedings of the 15th annual ACM Symposium on Parallel Algorithms and Architectures, pp. 34–43 (2003)

Harrison, J.M.: Brownian models of queueing networks with heterogenous customer populations. In: Fleming, W., Lions, P.L. (eds.) Stochastic Differential Systems, Stochastic Control Theory and Applications. Proceedings of the International Mathematics Association, pp. 147–186. Springer, New York (1988)

Harrow, A., Hayden, P., Leung, D.: Superdense coding of quantum states. Phys. Rev. Lett. **92**, 187901 (2004)

Harrow, A.W.: Coherent communication of classical messages. Phys. Rev. Lett. **92**, 097902 (2004)

Hartline, J., McGrew, R.: From optimal limited to unlimited supply auctions. In: Proc. of the 7th ACM Conference on Electronic Commerce (EC'05), 2005

Hartman, T., Shamir, R.: A simpler and faster 1.5-approximation algorithm for sorting by transpositions. Inf. Comput. **204**, 275–290 (2006)

Hartman, T., Sharan, R.: A 1.5-approximation algorithm for sorting by transpositions and transreversals. In: Proceedings of the 4th Workshop on Algorithms in Bioinformatics (WABI'04), pp. 50–61. Bergen, Norway, 17–21 Sep (2004)

Harvey, N.: Algebraic Structures and Algorithms for Matching and Matroid Problems. In: Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS), 2006

Harvey, N.J.A., Jones, M.B., Saroiu, S., Theimer, M., Wolman, A.: Skipnet: A scalable overlay network with practical locality properties. In: Proceedings of Fourth USENIX Symposium on Internet Technologies and Systems (USITS '03), March 2003

Håstad, J.: A slight sharpening of LMN. J. Comput. Syst. Sci. **63**(3), 498–508 (2001)

Håstad, J.: Clique is hard to approximate within $n^{1-\varepsilon}$. Acta Math. **182**(1), 105–142 (1999)

Hastad, J.: Some optimal inapproximability results. J. ACM **48**(4), 798–859 (2001)

Hastad, J., Wigderson, A.: Simple analysis of graph tests for linearity and pcp. Random Struct. Algorithms **22**(2), 139–160 (2003)

Hastie, T., Rosset, S., Tibshirani, R., Zhu, J.: The entire regularization path for the support vector machine. J. Mach. Learn. Res. **5**, 1391–1415 (2004)

Haussler, D.: Applying valiants learning framework to ai concept learning problems. In: Michalski, R., Kodratoff, Y. (eds.) Machine Learning: An Artificial Intelligence Approach. Morgan Kaufmann

Haussler, D.: Decision theoretic generalizations of the PAC model for neural net and other learning applications. Inf. Comput. **100**(1), 78–150 (1992)

Haussler, D.: Probably approximately correct learning and decision-theoretic generalizations. In: Smolensky, P., Mozer, M., Rumelhart, D. (eds.) Mathematical Perspectives on Neural Networks, pp. 651–718. L. Erlbaum Associates, Mahwah, New Jersey (1996)

Haveliwala, T., Kamvar, S., Jeh, G.: An Analytical Comparison of Approaches to Personalizing PageRank. In: Technical Report. Stanford University, Stanford (2003)

Haynes, T.W., Hedetniemi, S.T., Slater, P.J.: Domination in Graphs: Advanced Topics. Pure and Applied Mathematics, vol. 209. Marcel Dekker, New York (1998)

Haynes, T.W., Hedetniemi, S.T., Slater, P.J.: Fundamentals of Domination in Graphs. Pure and Applied Mathematics, vol. 208. Marcel Dekker, New York (1998)

Hayrapetyan, A., Swamy, C., Tardos, É.: Network design for information networks. In: SODA '05: Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms, Philadelphia, PA, USA, Society for Industrial and Applied Mathematics, pp. 933–942. (2005)

Hazay, C., Lewenstein, M., Sokol, D.: Approximate parameterized matching. ACM Trans. Algorithms **3**(3) (2007)

Hazay, C., Lewenstein, M., Tsur, D.: Two dimensional parameterized matching. In: Proc. of 16th Symposium on Combinatorial Pattern Matching (CPM), 2005, pp. 266–279

H.B.H. III, Marathe, M.V., Radhakrishnan, V., Ravi, S.S., Rosenkrantz, D.J., Stearns, R.E.: NC-approximation schemes for NP- and PSPACE-hard problems for geometric graphs. J. Algorithms **26**(2), 238–274 (1998)

He, M., Munro, J.I., Rao, S.S.: Succinct ordinal trees based on tree covering. In: Proc. 34th International Colloquium on Algorithms, Language and Programming (ICALP). LNCS n. 4596, pp. 509–520. Springer, Wroclaw, Poland (2007)

He, Y.-J., Huynh, T.N.D., Jannson, J., Sung, W.-K.: Inferring phylogenetic relationships avoiding forbidden rooted triplets. J Bioinform. Comput. Biol. **4**(1), 59–74 (2006)

Hedetniemi, S.M., Hedetniemi, S.T., Liestman, A.: A survey of gossiping and broadcasting in communication networks. Networks **18**, 129–134 (1988)

Hegedüs, T.: Generalized teaching dimensions and the query complexity of learning. In: COLT '95 Proceedings of the 8th Annual Conference on Computational Learning Theory, pp. 108–117 (1995)

Heggernes, P., Telle, J.A., Villanger, Y.: Computing minimal triangulations in time $O(n^{\alpha} \log n) = o(n^{2.376})$. SIAM J. Discret. Math. **19**(4), 900–913 (2005)

Hein, J.: A heuristic method to reconstruct the history of sequences subject to recombination. J. Mol. Evol. **36**, 396–405 (1993)

Hein, J.: An optimal algorithm to reconstruct trees from additive distance data. Bull. Math. Biol. **51**, 597–603 (1989)

Hein, J.: Reconstructing evolution of sequences subject to recombination using parsimony. Math. Biosci. **98**(2), 185–200 (1990)

Hein, J., Jensen, J., Pedersen, C.: Recursions for statistical multiple alignment. PNAS **100**, 14,960–14,965 (2003)

Hein, J., Jiang, T., Wang, L., Zhang, K.: On the complexity of comparing evolutionary trees. Discrete Appl. Math. **71**(1–3), 153–169 (1996)

Heinzelman, W., Chandrakasan, A., Balakrishnan, H.: Energy-efficient communication protocol for wireless microsensor networks. In: Proceedings of the 33rd IEEE Hawaii International Conference on System Sciences (HICSS 2000). 2000

Held, M.: VRONI: An engineering approach to the reliable and efficient computation of Voronoi diagrams of points and line segments. Comput. Geom. Theor. Appl. **18**(2), 95–123 (2001)

Held, M., Karp, R.M.: The traveling salesman problem and minimum spanning trees. Oper. Res. **18**, 1138–1162 (1970)

Hellerstein, L., Pillaipakkamnatt, K., Raghavan, V., Wilkins, D.: How many queries are needed to learn? J. ACM. **43**(5), 840–862 (1996)

Hellerstein, L., Raghavan, V.: Exact learning of dnf formulas using dnf hypotheses. J Comput. Syst. Sci. **70**(4), 435–470 (2005)

Helman, D.R., JáJá, J.: Sorting on clusters of SMP's. In: Proc. 12th Int'l Parallel Processing Symp., pp. 1–7, Orlando, FL, March/April 1998

Helmbold, D., Mayr, E.: Two processor scheduling is in NC. SIAM J. Comput. **16**(4), 747–756 (1987)

Helmuth, L.: Genome research: Map of the human genome 3.0. Science **293**(5530), 583–585 (2001)

Helsgaun, K.: An effective implementation of the Lin-Kernighan traveling salesman heuristic. Eur. J. Oper. Res. **126**(1), 106–130 (2000)

Hennessy, J.L., Patterson, D.A.: Computer Architeture: A Quantitative Approach, 2nd edn. Morgan Kaufmann, San Francisco, CA (1996)

Henzinger, M., King, V.: Fully dynamic biconnectivity and transitive closure. In: Proc. 36th IEEE Symposium on Foundations of Computer Science (FOCS'95). IEEE Computer Society, pp. 664–672. Los Alamos (1995)

Henzinger, M.R.: Fully dynamic biconnectivity in graphs. Algorithmica **13**(6), 503–538 (1995)

Henzinger, M.R.: Improved data structures for fully dynamic biconnectivity. SIAM J. Comput. **29**(6), 1761–1815 (2000)

Henzinger, M.R., Fredman, M.L.: Lower bounds for fully dynamic connectivity problems in graphs. Algorithmica **22**(3), 351–362 (1998)

Henzinger, M.R., King, V.: Maintaining minimum spanning forests in dynamic graphs. SIAM. J. Comput. **31**(2), 364–374 (2001)

Henzinger, M.R., King, V.: Randomized fully dynamic graph algorithms with polylogarihmic time per operation. In: Proceedings of the 27th Annual ACM Symposium on Theory of Computing (STOC), pp. 519–527 (1997)

Henzinger, M.R., King, V.: Randomized fully dynamic graph algorithms with polylogarithmic time per operation. J. ACM **46**(4), 502–516 (1999)

Henzinger, M.R., King, V., Warnow, T.: Constructing a tree from homeomorphic subtrees, with applications to computational evolutionary biology. Algorithmica **24**(1), 1–13 (1999)

Henzinger, M.R., Klein, P.N., Rao, S., Subramanian, S.: Faster Shortest-Path Algorithms for Planar Graphs. J. Comput. Syst. Sci. **55**, 3–23 (1997)

Henzinger, M.R., Thorup, M.: Sampling to provide or to bound: With applications to fully dynamic graph algorithms. Random Struct. Algorithms **11**(4), 369–379 (1997) (presented at ICALP 1996)

Herlihy, M.: A methodology for implementing highly concurrent data objects. ACM Trans. Program. Lang. Syst. **15**(5), 745–770 (1993)

Herlihy, M.: Wait-free synchronization. ACM Trans. Program. Lang. Syst. (TOPLAS) **13**(1), 124–149 (1991)

Herlihy, M., Luchangco, V., Moir, M.: Obstruction-free mechanism for atomic update of multiple non-contiguous locations in shared memory. US Patent Application 20040034673 (2002)

Herlihy, M., Luchangco, V., Moir, M.: Obstruction-free synchronization: Double-ended queues as an example. In: Proceedings of the 23rd International Conference on Distributed Computing Systems, 2003

Herlihy, M., Luchangco, V., Moir, M., Scherer III., W.: Software transactional memory for supporting dynamic-sized data structures. In: Proc. 22th Annual ACM Symposium on Principles of Distributed Computing, 2003, pp. 92–101

Herlihy, M., Moss, J.E.B.: Transactional memory: Architectural support for lock-free data structures. In: Proc. 20th Annual International Symposium on Computer Architecture, 1993, pp. 289–300

Herlihy, M., Rajsbaum, S.: A classification of wait-free loop agreement tasks. Theor. Comput. Sci. **291**(1), 55–77 (2003)

Herlihy, M., Rajsbaum, S.: Algebraic spans. Math. Struct. Comput. Sci. **10**(4), 549–573 (2000)

Herlihy, M., Rajsbaum, S.: Set consensus using arbitrary objects. In: Proceedings of the 13th Annual ACM Symposium on Principles of Distributed Computing, pp. 324–333, August (1994)

Herlihy, M., Rajsbaum, S.: The decidability of distributed decision tasks (extended abstract). In: STOC '97: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing, pp. 589–598. ACM Press, New York (1997)

Herlihy, M., Rajsbaum, S., Tuttle, M.R.: Unifying synchronous and asynchronous message-passing models. In: PODC '98: Proceedings of the seventeenth annual ACM symposium on Principles of distributed computing, pp. 133–142. ACM Press, New York (1998)

Herlihy, M.P., Penso, L.D.: Tight Bounds for *k*-Set Agreement with Limited Scope Accuracy Failure Detectors. Distrib. Comput. **18**(2), 157–166 (2005)

Herlihy, M.P., Shavit, N.: The asynchronous computability theorem for *t*-resilient tasks. In: Proceedings 25th Annual ACM Symposium on Theory of Computing, 1993, pp. 111–120

Herlihy, M.P., Shavit, N.: The Topological Structure of Asynchronous Computability. J. ACM **46**(6), 858–923 (1999)

Herlihy, M.P., Wing, J.M.: Linearizability: a correctness condition for concurrent objects. ACM Trans. Program. Lang. Syst. (TOPLAS) **12**(3), 463–492 (1990)

Hershberger, J., Suri, M.R., Suri, S.: Data structures for two-edge connectivity in planar graphs. Theor. Comput. Sci. **130**(1), 139–161 (1994)

Hertz, G., Stormo, G.: Identification of consensus patterns in unaligned DNA and protein sequences: a large-deviation statistical basis for penalizing gaps. In: Proc. 3rd Int'l Conf. Bioinformatics and Genome Research, pp. 201–216. (1995)

Herzog, S., Shenker, S., Estrin, D.: Sharing the "cost" of multicast trees: an axiomatic analysis. IEEE/ACM Trans. Netw. **5**, 847–860 (1997)

High Performance Fortran Forum. High Performance Fortran Language Specification, 1.0 edition, May 1993

Hillar, C.J., Rhea, D.L. A Result about the Density of Iterated Line Intersections. Comput. Geom.: Theory Appl. **33**(3), 106–114 (2006)

Hipke, C., Icking, C., Klein, R., Langetepe, E.: How to find a point on a line within a fixed distance. Discret. Appl. Math. **93**, 67–73 (1999)

Hirsch, E.A.: A $2^{m/4}$-time Algorithm for Max 2-SAT: Corrected Version. Electronic Colloquium on Computational Complexity Report TR99-036 (2000)

Hirsch, E.A.: New worst-case upper bounds for SAT. J. Autom. Reason. **24**(4), 397–420 (2000)

Hjelle, Ø., Dæhlen, M.: Triangulations and Applications. In: Mathematics and Visualization, vol. IX. Springer, Heidelberg (2006). ISBN 978-3-540-33260-2

Ho, J.M., Vijayan, G., Wong, C.K.: New algorithms for the rectilinear steiner tree problem. IEEE Transac. Comput. Aided Des. **9**, 185–193 (1990)

Hoang, V.T., Sung, W.K.: Fixed Parameter Polynomial Time Algorithms for Maximum Agreement and Compatible Supertrees. In: Albers, S., Weil, P., 25th International Symposium on Theoretical Aspects of Computer Science (STACS 2008). Dagstuhl, Germany (2007)

Hoare, C.A.R.: Quicksort. Comput. J. **5**(1), 10–15 (1962)

Hochbaum, D.S.: Approximation algorithms for the set covering and vertex cover problems. SIAM J. Comput. **11**(3), 555–556 (1982)

Hochbaum, D.S.: Heuristics for the fixed cost median problem. Math. Program. **22**(2), 148–162 (1982)

Hochbaum, D.S., Maass, W.: Approximation schemes for covering and packing problems in image processing and VLSI. J. ACM **32**(1), 130–136 (1985)

Hochbaum, D.S., Shmoys, D.B.: A best possible approximation algorithm for the *k*-center problem. Math. Oper. Res. **10**, 180–184 (1985)

Hochbaum, D.S., Shmoys, D.B.: A polynomial approximation scheme for scheduling on uniform processors: using the dual approximation approach. SIAM J. Comput. **17**(3), 539–551 (1988)

Hochbaum, D.S., Shmoys, D.B.: Using dual approximation algorithms for scheduling problems: theoretical and practical results. J. ACM **34**(1), 144–162 (1987)

Hoefer, M.: Experimental comparison of heuristic and approximation algorithms for uncapacitated facility location. In: Proceedings of the 2nd International Workshop on Experimental and Efficient Algorithms (WEA). Lecture Notes in Computer Science, vol. 2647, pp. 165–178. Springer, Berlin (2003)

Hoeffding, W.: On the distribution of the number of successes in independent trials. Ann. Math. Stat. **27**, 713–721 (1956)

Hofacker, I.L., Stadler, P.F.: Memory efficient folding algorithms for circular RNA secondary structures. Bioinformatics **22**, 1172–1176 (2006)

Hoffmann, F., Icking, C., Klein, R., Kriegel, K.: The polygon exploration problem. SIAM J. Comput. **31**(2), 577–600 (2001)

Hoffmann, M., Okamoto, Y.: The minimum weight triangulation problem with few inner points. Comput. Geom. Theory Appl. **34**, 149–158 (2006)

Hofmeister, T., Schöning, U., Schuler, R., Watanabe, O.: A probabilistic 3–SAT algorithm further improved. In: STACS 2002. LNCS, vol. 2285, pp. 192–202. Springer, Berlin (2002)

Hofmeister, T., Schöning, U., Schuler, R., Watanabe, O.: Probabilistic 3-SAT algorithm further improved. Proceedings 19th Symposium on Theoretical Aspects of Computer Science. LNCS **2285**, 193–202 (2002)

Hofri, M.: A feedback-less distributed broadcast algorithm for multihop radio networks with time-varying structure. In: Computer Performance and Reliability, pp. 353–368. (1987)

Holevo, A.S.: Bounds for the quantity of information transmitted by a quantum communication channel. Problemy Peredachi Informatsii, **9**, 3–11 (1973). English translation in: Probl. Inf. Transm. **9**, 177–183 (1973)

Holm, J., de Lichtenberg, K., Thorup, M.: Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. J. ACM **48**, 723–760 (2001)

Holmes, I.: Using guide trees to construct multiple-sequence evolutionary hmms. Bioinform. **19**, i147–i157 (2003)

Holmes, I., Bruno, W.J.: Evolutionary HMMs: a Bayesian approach to multiple alignment. Bioinform. **17**(9), 803–820 (2001)

Holzer, M., Schulz, F., Wagner, D.: Engineering Multi-Level Overlay Graphs for Shortest-Path Queries. In: Algorithm Engineering and Experiments – ALENEX (SIAM, 2006), pp. 156–170. SIAM, Philadelphia (2006)

Hon, W., Sadakane, K., Sung, W.: Breaking a time-and-space barrier in constructing full-text indices. In: Proc. of the 44th IEEE Symposium on Foundations of Computer Science (FOCS), 251–260, Cambridge, MA (2003)

Hon, W.K., Kao, M.Y., Lam, T.W., Sung, W.K., Yiu, S.M.: Non-shared Edges and Nearest Neighbor Interchanges revisited. Inf. Process. Lett. **91**(3), 129–134 (2004)

Hon, W.K., Lam, T.W.: Approximating the Nearest Neighbor Intercharge Distance for Non-Uniform-Degree Evolutionary Trees. Int. J. Found. Comp. Sci. **12**(4), 533–550 (2001)

Hon, W.K., Lam, T.W., Yiu, S.M., Kao, M.Y., Sung, W.K.: Subtree Transfer Distance For Degree-D Phylogenies. Int. J. Found. Comp. Sci. **15**(6), 893–909 (2004)

Hong, X., Dong, S., Ma, Y., Cai, Y., Cheng, C.K., Gu, J.: Corner Block List: An efficient topological representation of non-slicing floorplan. In: International Computer Aided Design (ICCAD) '00, November 2000, pp. 8–12,

Hoot, S.B., Palmer, J.D.: Structural rearrangements, including parallel inversions, within the chloroplast genome of Anemone and related genera. J. Mol. Evol. **38**, 274–281 (1994)

Hopcroft, J., Tarjan, R.: Efficient planarity testing. J. ACM **21**, pp. 549–568 (1974)

Hopcroft, J., Ullman, J.: Introduction to Automata, Languages, and Computation. Addison-Wesley, Reading, MA (1979)

Hopcroft, J.E., Karp, R.M.: An $O(n^{5/2})$ Algorithm for Maximum Matchings in Bipartite Graphs. SIAM J. Comput. **2**, 225–231 (1973)

Horn, W.: Minimizing average flow time with parallel machines. Oper. Res. **21**, 846–847 (1973)

Horton, J.D.: A Polynomial-time algorithm to find the shortest cycle basis of a graph. SIAM J. Comput. **16**(2), 358–366 (1987)

Hou, T., Li, V.: Transmission Range Control in Multihop Packet Radio Networks. IEEE Tran. Commun. **34**, 38–44 (1986)

Howard, P.G., Vitter, J.S.: Fast and efficient lossless image compression. In: Proceedings of the IEEE Data Compression Conference, Snowbird, Utah, March 1993, pp. 351–360

Howard, P.G., Vitter, J.S.: Parallel lossless image compression using Huffman and arithmetic coding. In: Proceedings of the IEEE Data Compression Conference, Snowbird, Utah, March 1992, pp. 299–308

Howard, P.G., Vitter, J.S.: Practical implementations of arithmetic coding. In: Storer, J.A. (ed.) Images and Text Compression. Kluwer Academic Publishers, Norwell, Massachusetts (1992)

Høyer, P.: Conjugated operators in quantum algorithms. Phys. Rev. A **59**(5), 3280–3289 (1999)

Høyer, P., Mosca, M., de Wolf, R.: Quantum search on bounded-error inputs. In: Proceedings of the 30th International Colloquium on Automata, Languages and Programming. Lecture Notes in Computer Science, vol. 2719, pp. 291–299, Eindhoven, The Netherlands, 30 June – 4 July 2003

Hromkovic, J., Klasing, R., Monien, B., Peine, R.: Dissemination of information in interconnection networks (broadcasting and gossiping). In: Du, D.Z., Hsu, F. (eds.) Combinatorial Network Theory, pp. 125–212. Kluwer Academic Publishers, Dordrecht (1996)

Hsu, W.L., McConnell, R.M.: PC trees and circular-ones arrangements. Theor. Comput. Sci. **296**(1), 99–116 (2003)

http://www.carms.ca (Canadian Resident Matching Service website)

http://www.jrmp.jp (Japan Resident Matching Program website)

http://www.nes.scot.nhs.uk/sfas/ (Scottish Foundation Allocation Scheme website)

http://www.nrmp.org/ (National Resident Matching Program website)

Hu, B., Marek-Sadowska, M.: Multilevel fixed-point-addition-based VLSI placement. IEEE Trans. CAD **24**(8), 1188–1203 (2005)

Hu, T.C.: Multi-commodity network flows. Operations Research, **11**(3), 344–360. (1963)

Hu, T.C., Moerder, K.: Multiterminal Flows in a Hypergraph. In: Hu, T.C., Kuh, E.S. (eds.) VLSI Circuit Layout: Theory and Design, pp. 87–93. IEEE Press (1985)

Huang, X.: An algorithm for identifying regions of a DNA sequence that satisfy a content requirement. Comput. Appl. Biosci. **10**, 219–225 (1994)

Huang, X., Pan, V.Y.: Fast rectangular matrix multiplications and applications. J. Complex. **14**, 257–299 (1998)

Huddleston, S., Mehlhorn, K.: A new data structure for representing sorted lists. Acta Inform. **17**, 157–184 (1982)

Hudson, R.: Gene genealogies and the coalescent process. Oxf. Surv. Evol. Biol. **7**, 1–44 (1990)

Hudson, R.: Generating samples under the wright-fisher neutral model of genetic variation. Bioinformatics **18**(2), 337–338 (2002)

Huffman, D.A.: A method for the construction of minimum redundancy codes. Proceedings of the Institute of Radio Engineers, 40, pp. 1098–1101 (1952)

Hüffner, F.: Graph Modification Problems and Automated Search Tree Generation. Diplomarbeit, Wilhelm-Schickard-Institut für Informatik, Universität Tübingen (2003)

Hüffner, F., Wernicke, S., Zichner, T.: Algorithm engineering for Color Coding to facilitate Signaling Pathway Detection. In: Proceedings of the 5th Asia-Pacific Bioinformatics Conference (APBC), pp. 277–286 (2007)

Hunt, E., Atkinson, M., Irving, R.: Database indexing for large DNA and protein sequence collections. Int. J. Very Large Data Bases **11**, 256–271 (2002)

Husfeldt, T., Rauhe, T.: New lower bound techniques for dynamic partial sums and related problems. SIAM J. Comput. **32**, 736–753 (2003). See also ICALP'98

Huson, D.H., Bryant, D.: Application of phylogenetic networks in evolutionary studies. Mol. Biol. Evol. **23**(2), 254–267 (2006)

Huson, D.H., Nettles, S., Warnow, T.: Disk-covering, a fast-converging method for phylogenetic tree reconstruction. J. Comput. Biol. **6**, 369–386 (1999)

Huson, D.H., Nettles, S., Warnow, T.: Obtaining highly accurate topology estimates of evolutionary trees from very short sequences. In: RECOMB, 1999, pp. 198–207

Hutchinson, D.A., Sanders, P., Vitter, J.S.: Duality between prefetching and queued writing with parallel disks. SIAM J. Comput. **34**, 1443–1463 (2005)

Huynh, T.N.D., Hon, W.K., Lam, T.W., Sung, W.K.: Approximate string matching using compressed suffix arrays. In: Proceedings of Symposium on Combinatorial Pattern Matching, 2004, pp. 434–444

Hwang, F.K.: On Steiner minimal trees with rectilinear distance. SIAM J. Appl. Math. **30**, 104–114 (1976)

Hwang, F.K., Richards, D.S., Winter, P.: The Steiner Tree Problem. North-Holland, Amsterdam (1992)

Hylland, A., Zeeckhauser, R.: The efficent allocation of individuals to positions. J. Polit. Econ. **87**(2), 293–314 (1979)

Iacono, J.: Key-independent optimality. Algorithmica **42**(1), 3–10 (2005)

Ibarra, O.H., Kim, C.E.: Fast approximation algorithms for the knapsack and sum of subset problem. J. ACM **22**, 463–468 (1975)

Idury, R.M., Schäffer, A.A.: Dynamic dictionary matching with failure functions. In: Proc. 3rd Annual Symposium on Combinatorial Pattern Matching, 1992, pp. 273–284

Idury, R.M., Schäffer, A.A.: Multiple matching of parametrized patterns. Theor. Comput. Sci. **154**(2), 203–224 (1996)

Ieong, S., Kao, M.-Y., Lam, T.-W., Sung, W.-K., Yiu, S.-M.: Predicting RNA secondary structures with arbitrary pseudoknots by maximizing the number of stacking pairs. In: Proceedings of the 2nd Symposium on Bioinformatics and Bioengineering, 2001, pp. 183–190

Ilie, L.: A simple proof that a word of length n has at most 2n distinct squares. J. Combin. Theory, Ser. A **112**(1), 163–164 (2005)

Ilie, L., Navarro, G., Yu, S.: On NFA reductions. In: Karhumäki, J. et al. (eds.) Theory is Forever. Lect. Notes Comput. Sci. **3113**, 112–124 (2004)

Iliopoulos, C., Moore, D., Smyth, W.F.: A characterization of the squares in a Fibonacci string. Theor. Comput. Sci. **172** 281–291 (1997)

Imahori, S.: Privatre communication, December 2005

Iman, S., Pedram, M., Fabian, C., Cong, J.: Finding Uni-Directional Cuts Based on Physical Partitioning and Logic Restructuring. In: 4th ACM/SIGDA Physical Design Workshop, April 1993

Impagliazzo, R., Paturi, R.: Which problems have strongly exponential complexity? J. Comput. Syst. Sci. **63**, 512–530 (2001)

Indyk, P.: Explicit constructions of selectors and related combinatorial structures, with applications. In: Proc. 13th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 697–704, San Francisco, USA (2002)

Indyk, P., Matousek, J.: Low-distortion embeddings of finite metric spaces. In: Goodman, J.E., O'Rourke, J. (eds.) Handbook of Discrete and Computational Geometry. CRC Press, Inc., Chap. 8 (2004), To appear

Indyk, P., Motwani, R.: Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. Proc. ACM STOC 604–613 (1998)

Intanagonwiwat, C., Govindan, R., Estrin, D.: Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks. In: Proc. 6th ACM/IEEE International Conference on Mobile Computing, MOBICOM'2000

Ion, A., Kropatsch, W.G., Haxhimusa, Y.: Considerations regarding the minimum spanning tree pyramid segmentation method. In: Proc. 11th Workshop Structural, Syntactic, and Statistical Pattern Recognition (SSPR). LNCS, vol. 4109, pp. 182–190. Springer, Berlin (2006)

Irani, S.: Page replacement with multi-size pages and applications to Web caching. Algorithmica **33**(3), 384–409 (2002)

Irani, S.: Two results on the list update problem. Inf. Proc. Lett. **38**, 301–306 (1991)

Irani, S., Karlin, A.R., Phillips, S.: Strongly competitive algorithms for paging with locality of reference. SIAM J. Comput. **25**(3), 477–497 (1996)

Irani, S., Pruhs, K.: Algorithmic Problems in Power Management. ACM SIGACT News **36**(2), 63–76. New York, NY, USA (2005)

Irani, S., Seiden, S.S.: Randomized algorithms for metrical task systems. Theor. Comput. Sci. **194**, 163–182 (1998)

Irving, R.W.: An efficient algorithm for the stable roommates problem. J. Algorithms **6**, 577–595 (1985)

Irving, R.W.: Matching medical students to pairs of hospitals: a new variation on a well-known theme. Proc. ESA 98. LNCS 1461, pp. 381–392. (1998)

Irving, R.W.: Stable marriage and indifference. Discret. Appl. Math. **48**, 261–272 (1994)

Irving, R.W., Kavitha, T., Mehlhorn, K., Michail, D., Paluch, K.: Rank-maximal matchings. In: Proceedings of the 15th ACM-SIAM Symposium on Discrete Algorithms, pp. 68–75. SIAM, New Orleans (2004)

Irving, R.W., Leather, P.: The complexity of counting stable marriages. SIAM J. Comput. **15**(3), 655–667 (1986)

Irving, R.W., Leather, P., Gusfield, D.: An efficient algorithm for the "optimal stable" marriage. J. ACM **34**(3), 532–543 (1987)

Irving, R.W., Manlove, D.F.: The stable roommates problem with ties. J. Algorithms **43**, 85–105 (2002)

Irving, R.W., Manlove, D.F., O'Malley, G.: Stable marriage with ties and bounded length preference lists. Proc. the 2nd Algorithms and Complexity in Durham workshop, Texts in Algorithmics, College Publications (2006)

Irving, R.W., Manlove, D.F., Scott, S.: Strong stability in the Hospitals/Residents problem. In: Proceedings of STACS 2003: the 20th Annual Symposium on Theoretical Aspects of Computer Science. Lecture Notes in Computer Science, vol. 2607, pp. 439–450. Springer, Berlin (2003)

Irving, R.W., Manlove, D.F., Scott, S.: The Hospitals/Residents problem with Ties. In: Proceedings of SWAT 2000: the 7th Scandinavian Workshop on Algorithm Theory. Lecture Notes in Computer Science, vol. 1851, pp. 259–271. Springer, Berlin (2000)

Irving, R.W., Scott, S.: The stable fixtures problem – a many-to-many extension of stable roommates. Discret. Appl. Math. **155**, 2118–2129 (2007)

Ismailescu, D., Radoičić, R.: A Dense Planar Point Set from Iterated Line Intersections. Comput. Geom. Theory Appl. **27**(3), 257–267 (2004)

Israeli, A., Jalfon, M.: Token Management Schemes and Random Walks Yield Self-Stabilizing Mutual Exclusion. In: Proceedings of the 9th Annual ACM Symposium on Principles of Distributed Computing, pp. 119–131, Quebec City, August 1990

Israeli, A., Li, M.: Bounded time-stamps. Distrib. Comput. **6**(4), 205–209 (1993)

Israeli, A., Shaham, A.: Optimal multi-writer multireader atomic register. In: Proc. 11th ACM Symp. Principles Distr. Comput., pp. 71–82. Vancouver, British Columbia, Canada, 10–12 August 1992

Itai, A., Konheim, A.G., Rodeh, M.: A sparse table implementation of priority queues. In: Automata, Languages and Programming, 8th Colloquium. LNCS, vol. 115, pp. 417–431. Springer, Berlin (1981)

Itai, A., Rodeh, M.: Finding a Minimum Circuit in a Graph. SIAM J. Comput. **7**(4), 413–423 (1978)

Italiano, G.F., La Poutré, J.A., Rauch, M.: Fully dynamic planarity testing in planar embedded graphs. 1st Annual European Symposium on Algorithms, Bad Honnef, Germany, 30 September–2 October 1993

Iwama, K., Manlove, D.F., Miyazaki, S., Morita, Y.: Stable marriage with incomplete lists and ties. Proc. ICALP 99. LNCS 1644, pp. 443–452. (1999)

Iwama, K., Miyazaki, S., Yamauchi, N.: A 1.875-approximation algorithm for the stable marriage problem. Proc, SODA 2007, pp. 288–297. (2007)

Iwama, K., Tamaki, S.: Improved upper bounds for 3-SAT. In: Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms, 2004, pp. 328–329

Iwasaki, A., Yokoo, M., Terada, K.: A robust open ascending-price multi-unit auction protocol against false-name bids. Decis. Support. Syst. **39**, 23–39 (2005)

Iyer, R., Karger, D., Rahul, H., Thorup, M.: An experimental study of poly-logarithmic fully-dynamic connectivity algorithms. J. Exp. Algorithmics **6**(4) (2001) (presented at ALENEX 2000)

Jaakkola, T.S., Haussler, D.: Probabilistic kernel regression models. In: Proceedings of the 1999 Conference on AI and Statistics Fort Lauderdale (1999)

Jackson, D.: Software Abstractions: Logic, Language, and Analysis. MIT Press (2006)

Jackson, J.: An efficient membership-query algorithm for learning DNF with respect to the uniform distribution. J. Comput. Syst. Sci. **55**, 414–440 (1997)

Jackson, J., Klivans, A., Servedio, R.: Learnability beyond AC^0. In: Proceedings of the 34th ACM Symposium on Theory of Computing, pp. 776–784, Montréal, 23–25 May 2002

Jackson, J., Shamir, E., Shwartzman, C.: Learning with queries corrupted by classification noise. Discret. Appl. Math. **92**, 157–175 (1999)

Jackson, J., Shamir, E., Shwartzman, C.: Learning with queries corrupted by classification noise. In: Proceedings of the Fifth Israel Symposium on the Theory of Computing Systems, pp. 45–53 (1997)

Jackson, J.C.: An efficient membership-query algorithm for learning DNF with respect to the uniform distribution. In: 35th Annual Symposium on Foundations of Computer Science, pp. 42–53. IEEE Computer Society Press, Los Alamitos (1994)

Jackson, J.C.: The Harmonic Sieve: A Novel Application of Fourier Analysis to Machine Learning Theory and Practice. Ph. D. thesis, Carnegie Mellon University (1995)

Jacobsen, L., Larsen, K.S., Nielsen, M.N.: On the existence of non-extreme (a,b)-trees. Inform. Process. Lett. **84**, 69–73 (2002)

Jacobson, G.: Space-efficient static trees and graphs. In: Proc. 30th IEEE Symposium on Foundations of Computer Science (FOCS), pp. 549–554. Triangle Park, USA (1989)

Jacobson, H., Stockmayer, W.H.: Intramolecular reaction in polycondensations. I. the theory of linear systems. J. Chem. Phys. **18**, 1600–1606 (1950)

Jaeger, F., Vertigan, D., Welsh, D.: On the computational complexity of the Jones and Tutte polynomials. Math. Proc. Cambridge Philos. Soc. **108**(1), 35–53 (1990)

Jaffe, J.: Bottleneck Flow Control. IEEE Trans. Commun. **29**(7), 954–962 (1981)

Jain, K.: A factor 2 approximation for the generalized Steiner network problem. Combinatorica **21**(1), 39–60 (2001)

Jain, K.: A polynomial time algorithm for computing the Arrow-Debreu market equilibrium for linear utilities. In: Proceeding of FOCS'04, pp. 286–294. IEEE Computer Society, Rome (2004)

Jain, K., Mahdian, M., Markakis, E., Saberi, A., Vazirani, V.V.: Approximation algorithms for facility location via dual fitting with factor-revealing LP. J. ACM **50**(6), 795–824 (2003)

Jain, K., Mahdian, M., Markakis, E., Saberi, A., Vazirani, V.V.: Greedy facility location algorithms analyzed using dual fitting with factor-revealing LP. J. ACM **50**(6), 795–824 (2003)

Jain, K., Mahdian, M., Saberi, A.: A new greedy approach for facility location problems. In: Proceedings of the 34st Annual ACM Symposium on Theory of Computing (STOC) pp. 731–740, ACM Press, New York (2002)

Jain, K., Padhye, J., Padmanabhan, V.N., Qiu, L.: Impact of interference on multi-hop wireless network performance. In: Proc. ACM MOBICOM 2003, pp. 66–80

Jain, K., Vazirani, V.V.: An approximation algorithm for the fault tolerant metric facility location problem. In: Approximation Algorithms for Combinatorial Optimization, Proceedings of APPROX (2000), vol. (1913) of Lecture Notes in Computer Science, pp. 177–183. Springer, Berlin (2000)

Jain, K., Vazirani, V.V.: Applications of approximation algorithms to cooperative games. In: Proc. of the 33rd Annual ACM Symposium on Theory of Computing, Association for Computing Machinery, New York, pp. 364–372 (2001)

Jain, K., Vazirani, V.V.: Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and lagrangian relaxation. J. ACM **48**(2), 274–296 (2001)

Jain, R.: Resource requirements of private quantum channels and consequence for oblivious remote state preparation. Technical report (2005). arXive:quant-ph/0507075

Jain, S., Shah, R., Brunette, W., Borriello, G., Roy, S.: Exploiting mobility for energy efficient data collection in wireless sensor networks. J. Mobile Netw. Appl. **11**(3), 327–339 (2006)

JáJá, J.: An Introduction to Parallel Algorithms. Addison-Wesley (1992)

Jamin, S., Jin, C., Jin, Y., Raz, D., Shavitt, Y., Zhang, L.: On the placement of internet instrumentations. In: Proceedings of the 19th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), vol. 1, pp. 295–304. IEEE Computer Society, Los Alamitos, CA, USA (2000)

Jampala, H., Zeh, N.: Cache-oblivious planar shortest paths. In: Proc. 32nd International Colloquium on Automata, Languages, and Programming. LNCS, vol. 3580, pp. 563–575. Springer, Berlin (2005)

Jannink, J.: Implementing deletions in B^+-trees. SIGMOD RECORD **24**, 33–38 (1995)

Jansen, K., Woeginger, G.J.: The complexity of detecting crossingfree configurations in the plane. BIT **33**, 580–595 (1993)

Janson, S.: Large Deviation Inequalities for Sums of Indicator Variables. Technical Report No. 34, Department of Mathematics, Uppsala University (1994)

Jansson, J., Joseph, H., Ng, K., Sadakane, K., Sung, W.-K.: Rooted maximum agreement supertrees. Algorithmica **43**(4), 293–307 (2005)

Jansson, J., Nguyen, N.B., Sung, W.-K.: Algorithms for combining rooted triplets into a galled phylogenetic network. SIAM J. Comput. **35**(5), 1098–1121 (2006)

Jansson, J., Sadakane, K., Sung, W.: Ultra-succinct representation of ordered trees. In: Proc. 18th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 575–584. New Orleans, USA (2007)

Jansson, J., Sung, W.-K.: Inferring a level-1 phylogenetic network from a dense set of rooted triplets. In: Proc. 10th International Computing and Combinatorics Conference (COCOON 2004), 2004

Japanese Resident Matching Program (JRMP) http://www.jrmp.jp/

Jarry, A., Leone, P., Powell, O., Rolim, J.: An Optimal Data Propagation Algorithm for Maximizing the Lifespan of Sensor Networks. In: Second International Conference, DCOSS 2006, San Francisco, CA, USA, June 2006. Lecture Notes in Computer Science, vol. 4026, pp. 405–421. Springer, Berlin (2006)

Jawor, W.: Three dozen papers on online algorithms. SIGACT News **36**(1), 71–85 (2005)

Jayanti, P.: An optimal multi-writer snapshot algorithm. In: Proc. 37th ACM Symposium on Theory of Computing. Baltimore, May 2005, pp. 723–732. ACM, New York (2005)

Jech, T.: The ranking of incomplete tournaments: A mathematician's guide to popular sports. Am. Math. Mon. **90**(4), 246–266 (1983)

Jeng-Fung, C.: Unrelated parallel machine scheduling with secondary resource constraints. Int. J. Adv. Manuf. Technol. **26**, 285–292 (2005)

Jermaine, C., Pol, A., Arumugam, S.: Online maintenance of very large random samples. In: SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data, New York, pp. 299–310. ACM Press (2004)

Jerrum, M.: A very simple Algorithm for Estimating the Number of k-colourings of a Low Degree Graph. Random Struct. Algorithms **7**, 157–165 (1994)

Jerrum, M.R., Valiant, L.G., Vazirani, V.V.: Random Generation of Combinatorial Structures from a Uniform Distribution. Theor. Comput. Sci. **43**, 169–188 (1986)

Jia, L., Rajaraman, R., Suel, R.: An Efficient Distributed Algorithm for Constructing Small Dominating Sets. In: PODC, Newport, Rhode Island, USA, August 2001

Jiang, M.: A 2-approximation for the preceding-and-crossing structured 2-interval pattern problem, J. Combin. Optim. **13**, 217–221 (2007)

Jiang, M.: A PTAS for the weighted 2-interval pattern problem over the preceding-and-crossing model. In: Y.X. A.W.M. Dress, B. Zhu (eds.) Proc. 1st Annual International Conference on Combinatorial Optimization and Applications (COCOA), Xi'an, China, Lecture Notes in Computer Science, vol. 4616, pp. 378–387. Springer (2007)

Jiang, M.: Improved approximation algorithms for predicting RNA secondary structures with arbitrary pseudoknots. In: Proc. 3rd International Conference on Algorithmic Aspects in Information and Management (AAIM), Portland, OR, USA, Lecture Notes in Computer Science, vol. 4508, pp. 399–410. Springer (2007)

Jiang, T., Kearney, P., Li, M.: A polynomial time approximation scheme for inferring evolutionary trees from quartet topologies and its application. SIAM J. Comput. **30**(6), 1942–1961 (2001)

Jiang, T., Ravikumar, B.: Minimal NFA problems are hard. SIAM J. Comput. **22**(6), 1117–1141 (1993)

Jiang, T., Wang, L., Zhang, K.: Alignment of trees – an alternative to tree edit. Theor. Comput. Sci. **143**(1), 137–148 (1995)

Joachims, T.: Text categorization with support vector machines. In: Proceedings of European Conference on Machine Learning (ECML) Chemnitz (1998)

Johansen, K.E., Jorgensen, U.L., Nielsen, S.H.: A distributed spanning tree algorithm. In: Proc. 2nd Int. Workshop on Distributed Algorithms (DISC). Lecture Notes in Computer Science, vol. 312, pp. 1–12. Springer, Berlin Heidelberg (1987)

Johansson, Ö.: Simple distributed $(\Delta + 1)$-coloring of graphs. Inf. Process. Lett. **70**, 229–232 (1999)

Johnson, D.: Efficient algorithms for shortest paths in sparse networks. J. Assoc. Comput. Mach. **24**, 1–13 (1977)

Johnson, D., Papadimitriou, C.H., Yannakakis, M.: How easy is local search? J. Comp. Syst. Sci. **37**, 79–100 (1988)

Johnson, D.B., Metaxas, P.: Connected Components in $O(\lg^{3/2} |V|)$ Parallel Time for the CREW PRAM. In: Proceedings of the 32nd Annual IEEE Symposium on Foundations of Computer Science, 1991, pp. 688–697

Johnson, D.B., Venkatesan, S.M.: Parallel algorithms for minimum cuts and maximum flows in planar networks. J. ACM **34**, 950–967 (1987)

Johnson, D.S.: A theoretician's guide to the experimental analysis of algorithms. In: Goodrich, M.H., Johnson, D.S., McGeoch, C.C. (eds.) Data Structures, Near Neighbors Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 59. American Mathematical Society, Providence (2002)

Johnson, D.S.: Approximation algorithms for combinatorial problems. J. Comput. Syst. Sci. **9**, 256–278 (1974)

Johnson, D.S.: Near-Optimal Bin Packing Algorithms. Ph. D. thesis, Massachusetts Institute of Technology, Department of Mathematics, Cambridge (1973)

Johnson, D.S., Demers, A., Ullman, J.D., Garey, M.R., Graham, R.L.: Worst-case performance bounds for simple one-dimensional packing algorithms. SIAM J. Comput. **3**, 299–325 (1974)

Johnson, D.S., Leighton, F.T., Shor, P.W., Weber, R.R.: The expected behavior of FFD, BFD, and optimal bin packing under $U(0, \alpha])$ distributions (in preparation)

Johnson, D.S., McGeoch, L.A.: Experimental analysis of heuristics for the STSP. In: Gutin, G., Punnen, A.P. (eds.) The Traveling Salesman Problem and its Variations. Kluwer, Dordrecht (2002)

Johnson, D.S., McGeoch, L.A.: The traveling salesman problem: A case study. In: Aarts, E., Lenstra, J.K. (eds.) Local Search in Combinatorial Optimization, pp. 215–310. Wiley, Chicester (1997)

Johnson, N.L., Kotz, S.: Urn Models and Their Applications. Wiley, New York (1977)

Johnson, W., Lindenstrauss, J.: Extensions of Lipschitz Mappings into a Hilbert Space. Contemp. Math. **26**, 189–206 (1984)

Jones, C.K.: A network model for foreign exchange arbitrage, hedging and speculation. Int. J. Theor. Appl. Finance **4**(6), 837–852 (2001)

Jones, V.F.R.: A polynomial invariant for knots via von Neumann algebras. Bull. Am. Math. Soc. **12**(1), 103–111 (1985)

Jordan, S., Shor, P.: Estimating Jones polynomials is a complete problem for one clean qubit. http://arxiv.org/abs/0707.2831 (2007)

Joseph, D., Meidanis, J., Tiwari, P.: Determining DNA sequence similarity using maximum independent set algorithms for interval graphs. In: Proc. 3rd Scandinavian Workshop on Algorithm Theory (SWAT). Lecture Notes in Computer Science, pp. 326–337. Springer, Berlin (1992)

Joswig, M.: Software. In: Goodman, J.E., O'Rourke, J. (eds.) Handbook of Discrete and Computational Geometry, 2nd edn., chap. 64, pp. 1415–1433. Chapman & Hall/CRC, Boca Raton (2004)

Jothi, R., Raghavachari, B., Varadarajan, S.: A 5/4-approximation algorithm for minimum 2-edge-connectivity. In: SODA, 2003, pp. 725–734

Joux, A., Stern, J.: Lattice reduction: A toolbox for the cryptanalyst. J. Cryptolo. **11**(3), 161–185 (1998)

Jozsa, R.: Notes on Hallgren's efficient quantum algorithm for solving Pell's equation, tech. report, quant-ph/0302134 (2003)

Jukes, T.H., Cantor, C.R.: Evolution of Protein Molecules. In: Munro, H.N. (ed.), Mammalian Protein Metabolism, pp. 21–132, Academic Press, New York (1969)

Jurdziński, T., Stachowiak, G.: Probabilistic Algorithms for the Wakeup Problem in Single-Hop Radio Networks. In: Proc. of the 13th Annual International Symposium on Algorithms and Computation (ISAAC), pp. 535–549 (2002)

Jutla, C., Patthak, A., Rudra, A., Zuckerman, D.: Testing low-degree polynomials over prime fields. In: Proceedings of the Forty-Fifth Annual Symposium on Foundations of Computer Science, pp. 423–432. IEEE, New York (2004)

Juurlink, B.H.H., Wijshoff, H.A.G.: A quantitative comparison of parallel computation models. ACM Trans. Comput. Syst. **13**(3), 271–318 (1998)

Kaashoek, F., Karger, D.R.: Koorde: A simple degree-optimal hash table. In: 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03), 2003

Kachirski, O., Guha, R.: Intrusion detection using mobile agents in wireless ad hoc networks. In: Proceedings of IEEE Workshop on Knowledge Media Networking, Kyoto, 10–12 July 2002

Kahn, J.M., Katz, R.H., Pister, K.S.J.: Next Century Challenges: Mobile Networking for Smart Dust. In: Proc. 5th ACM/IEEE International Conference on Mobile Computing, pp. 271–278, Sept. 1999

Kahng, A., Robins, G.: A new family of Steiner tree heuristics with good performance: the iterated 1-Steiner approach. In: Proceedings of IEEE Int. Conf. on Computer-Aided Design, Santa Clara, pp.428–431 (1990)

Kahng, A.B., Mandoiu, I.I., Zelikovsky, A.: Highly scalable algorithms for rectilinear and octilinear steiner trees. In: Proc. Asia and South Pacific Design Automation Conference, Kitakyushu, Japan, (2003) pp. 827–833

Kahng, A.B., Robins, G.: A new class of iterative steiner tree heuristics with good performance. IEEE Transac. Comput. Aided Des. **11**, 893–902 (1992)

Kahng, A.B., Wang, Q.: Implementation and extensibility of an analytic placer. IEEE Trans. CAD **24**(5), 734–747 (2005)

Kajitani, Y.: Theory of placement by Single-Sequence Realted with DAG, SP, BSG, and O-tree. In: International Symposium on Circuts and Systems, May 2006

Kakade, S.: On the Sample Complexity of Reinforcement Learning. Ph. D. thesis, University College London (2003)

Kaklamanis, C., Krizanc, D., Rao, S.: Hot-potato routing on processor arrays. In: Proceedings of the 5th Annual ACM, Symposium on Parallel Algorithms and Architectures, pp. 273–282, Velen (1993)

Kaklamanis, C., Krizanc, D., Tsantilas, T.: Tight bounds for oblivious routing in the hypercube. In: Proceedings of the 3rd annual ACM Symposium on Parallel Algorithms and Architectures, pp. 31–36 (1991)

Kalai, A., Klivans, A., Mansour, Y., Servedio, R.: Agnostically learning halfspaces. In: Proceedings of the 46th IEEE Symposium on Foundations of Computer Science (FOCS), pp. 11–20, Pittsburgh, PA, USA, 23–25 October 2005

Kalai, E., Zemel, E.: Generalized Network Problems Yielding Totally Balanced Games. Oper. Res. **30**, 998–1008 (1982)

Kalai, E., Zemel, E.: Totally Balanced Games and Games of Flow. Math. Oper. Res. **7**, 476–478 (1982)

Kallahalla, M., Varman, P.J.: Optimal read-once parallel disk scheduling. Algorithmica **43**, 309–343 (2005)

Kalyanasundaram, B., Pruhs, K.: Minimizing flow time nonclairvoyantly. In: Proceedings of the 38th Symposium on Foundations of Computer Science, October 1997

Kalyanasundaram, B., Pruhs, K.: Minimizing flow time nonclairvoyantly. J. ACM **50**(4), 551–567 (2003)

Kalyanasundaram, B., Pruhs, K.: Speed is as powerful as clairvoyance. In: Proceedings of the 36th Symposium on Foundations of Computer Science, October 1995, pp. 214–221

Kalyanasundaram, B., Pruhs, K.: Speed is as powerful as clairvoyance. J. ACM **47**(4), 617–643 (2000)

Kamath, A., Motwani, R., Spirakis, P., Palem, K.: Tail bounds for occupancy and the satisfiability threshold conjecture. J. Random Struct. Algorithms **7**(1), 59–80 (1995)

Kamel, I., Faloutsos, C.: Hilbert R-tree: An improved R-tree using fractals. In: Proc. International Conference on Very Large Databases, 1994, pp. 500–509

Kamel, I., Faloutsos, C.: On packing R-trees. In: Proc. International Conference on Information and Knowledge Management, 1993, pp. 490–499

Kamphans, T., Langetepe, E.: Optimal competitive online ray search with an error-prone robot. In: 4th International Workshop on Experimental and Efficient Algorithms, pp. 593–596 (2005)

Kanj, I.A., Nakhleh, L., Xia, G.: Reconstructing evolution of natural languages: Complexity and parametrized algorithms. In: Proceedings of the 12th Annual International Computing and Combinatorics Conference (COCOON 2006). Lecture Notes in Computer Science, vol. 4112, pp. 299–308. Springer, Berlin (2006)

Kannan, R.: Annual reviews of computer science, vol. 2, chap. "Algorithmic geometry of numbers", pp. 231–267. Annual Review Inc., Palo Alto, California (1987)

Kannan, R.: Minkowski's convex body theorem and integer programming. Math. Oper. Res. **12**(3), 415–440 (1987)

Kannan, R., Theobald, T.: Games of fixed rank: A hierarchy of bimatrix games. In: Proceedings of the ACM-SIAM Symposium on Discrete Algorithms, New Orleans, 7–9 January 2007

Kannan, S., Lawler, E., Warnow, T.: Determining the evolutionary tree using experiments. J. Algorithms **21**(1), 26–50 (1996)

Kannan, S., Sweedyk, Z., Mahaney, S.: Counting and random generation of strings in regular languages. In: *Proceedings of the 6th ACM-SIAM Symposium on Discrete Algorithms*, San Francisco, California, pp. 551–557. ACM Press, New York (1995)

Kannan, S., Warnow, T.: A fast algorithm for the computation and enumeration of perfect phylogenies. SIAM J. Comput. **26**, 1749–1763 (1997)

Kannan, S., Warnow, T.: Inferring evolutionary history from DNA sequences. SIAM J. Comput. **23**, 713–737 (1994)

Kanth, K.V.R., Singh, A.K.: Optimal dynamic range searching in non-replicating index structures. In: Proc. International Conference on Database Theory. LNCS, vol. 1540, pp. 257–276 (1999)

Kao, M.-Y.: Tree contractions and evolutionary trees. SIAM J. Comput. **27**(6), 1592–1616 (1998)

Kao, M.-Y., Lam, T.-W., Sung, W.-K., Ting, H.-F.: An even faster and more unifying algorithm for comparing trees via unbalanced bipartite matchings. J. Algorithms **40**(2), 212–233 (2001)

Kao, M.-Y., Li, X.-Y., Wang, W.: Output truthful versus input truthful: a new concept for algorithmic mechanism design (2006)

Kao, M.-Y., Li, X.-Y., Wang, W.: Towards truthful mechanisms for binary demand games: A general framework. In: ACM EC, pp. 213–222, Vancouver, Canada (2005)

Kao, M.-Y., Littman, M.L.: Algorithms for informed cows. In: AAAI-97 Workshop on On-Line Search, pp. 55–61 (1997)

Kao, M.-Y., Reif, J.H., Tate, S.R.: Searching in an unknown environment: an optimal randomized algorithm for the cow-path problem. Inf. Comput. **131**(1), 63–79 (1996) Preliminary version in SODA '93, pp. 441–447

Kao, M., Ma, Y., Sipser, M., Yin, Y.: Optimal constructions of hybrid algorithms. In: Proceedings 5th ACM-SIAM Symposium on Discrete Algorithms (SODA) pp. 372–381 (1994)

Kaplan, H., Landau, S., Verbin, E.: A simpler analysis of Burrows-Wheeler-based compression. Theoretical Computer Science **387**(3): 220–235 (2007)

Kaplan, H., Shafrir, N.: The greedy algorithm for shortest superstrings. Inform. Proc. Lett. **93**(1), 13–17 (2005)

Kaplan, H., Shamir, R., Tarjan, R.E.: Faster and simpler algorithm for sorting signed permutations by reversals. SIAM J. Comput. **29**, 880–892 (1999)

Kaplan, H., Verbin, E.: Efficient data structures and a new randomized approach for sorting signed permutations by reversals. In: Proceedings of the 14th Annual Symposium on Combinatorial Pattern Matching (CPM'03), pp. 170–185. Morelia, Michocán, Mexico, 25–27 Jun (2003)

Kaporis, A., Spirakis, P.G.: Stackelberg games on arbitrary networks and latency functions. In: 18th ACM Symposium on Parallelism in Algorithms and Architectures (2006)

Kaporis, A.C., Kirousis, L.M., Lalas, E.G.: The probabilistic analysis of a greedy satisfiability algorithm. Random Struct. Algorithms **28**(4), 444–480 (2006)

Karakostas, G.: Faster approximation schemes for fractional multicommodity flow problems. In: SODA '02: Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms, pp. 166–173. Society for Industrial and Applied Mathematics, Philadelphia (2002)

Karakostas, G., Kolliopoulos, G.: Stackelberg strategies for selfish routing in general multicommodity networks. Technical report, Advanced Optimization Laboratory, McMaster Univercity (2006) AdvOL2006/08, 2006-06-27

Karchmer, M.: Communication Complexity: A New Approach to Circuit Depth. MIT Press (1989)

Karger, D.: A Randomized Fully Polynomial Time Approximation Scheme for the All-Terminal Network Reliability Problem. SIAM J. Comput. **29**, 492–514 (1999)

Karger, D., Lehman, E., Leighton, F.T., Levine, M., Lewin, D., Panigrahy, R.: Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In: Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC), 1997, pp. 654–663 1997

Karger, D., Levine, M.: Random Sampling in Residual Graphs. In: Proc. of the 34th Annual ACM Symposium on Theory of Computing 2002, pp. 63–66

Karger, D., Minkoff, M.: Building Steiner trees with incomplete global knowledge. In: Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society, pp. 613–623. Los Alamitos (2000)

Karger, D., Motwani, R., Sudan, M.: Approximate graph coloring by semidefinite programming. J. ACM **45**(2), 246–265 (1998)

Karger, D.R.: Minimum cuts in near-linear time. J. ACM **47**(1), 46–76 (2000)

Karger, D.R.: Random sampling in Graph Optimization Problems. Ph. D. thesis, Department of Computer Science, Stanford University (1995)

Karger, D.R., Klein, P., Stein, C., Thorup, M., Young, N.E.: Rounding algorithms for a geometric embedding of minimum multiway cut. Math. Oper. Res. **29**(3), 436–461 (2004). Preliminary version in STOC 1999

Karger, D.R., Klein, P.N., Tarjan, R.E.: A randomized linear-time algorithm for finding minimum spanning trees. J. ACM **42**(2), 321–329 (1995)

Karger, D.R., Stein, C.: An $\tilde{O}(n^2)$ algorithm for minimum cut. In: Proceeding of 25th Annual ACM Symposium on Theory of Computing (STOC), 1993, pp. 757–765

Kärkkäinen, J.: Fast BWT in small space by blockwise suffix sorting. Theor. Comput. Sci. **387**, 249–257 (2007)

Kärkkäinen, J., Navarro, G., Ukkonen, E.: Approximate string matching on Ziv–Lempel compressed text. J. Discret. Algorithms **1**(3–4), 313–338 (2003)

Kärkkäinen, J., Sanders, P.: Simple linear work suffix array construction. In: Proceedings of the 30th International Colloquium on Automata, Languages, and Programming, ICALP 2003. LNCS, vol. 2719, pp. 943–955. Springer, Berlin (2003)

Kärkkäinen, J., Sanders, P., Burkhardt, S.: Linear work suffix array construction. J. ACM **53**(6), 918–936 (2006)

Kärkkäinen, J., Ukkonen, E.: Two- and higher-dimensional pattern matching in optimal expected time. SIAM J. Comput. **29**, 571–589 (1999)

Karl, H., Willig, A.: Protocols and Architectures for Wireless Sensor Networks. Wiley, West Sussex (2005)

Karlin, A., Yao, A.C.: Probabilistic lower bounds for the byzantine generals problem. Unpublished manuscript

Karlin, A.R., Manasse, M.S., McGeoch, L.A., Owicki, S.S.: Competitive Randomized Algorithms for Nonuniform Problems. Algorithmica **11**(6), 542–571 (1994) (Conference version: SODA 1990, pp. 301–309)

Karlin, A.R., Manasse, M.S., Rudolph, L., Sleator, D.D.: Competitive Snoopy Caching. Algorithmica **3**, 77–119 (1988) (Conference version: FOCS 1986, pp. 244–254)

Karlin, A.R., Phillips, S.J., Raghavan, P.: Markov paging. SIAM J. Comput. **30**(3), 906–922 (2000)

Karloff, H.: A Las Vegas RNC algorithm for maximum matching. Combinatorica **6**, 387–391 (1986)

Karloff, H., Rabani, Y., Ravid, Y.: Lower bounds for randomized k-server and motion-planning algorithms. SIAM J. Comput. **23**(2), 293–312 (1994)

Karloff, H.J.: How good is the Goemans-Williamson MAX CUT algorithm? SIAM J. Comput. **29**(1), 336–350 (1999)

Karmarkar, N.: A new polynomial-time algorithm for linear programming. Combinatorica **4**, 373–395 (1984)

Karmarkar, N., Karp, R.M.: An efficient approximation scheme for the one-dimensional bin packing problem. In: Proc. of the 23rd Annual Symposium on Foundations of Computer Science, pp. 312–320. IEEE Computer Soc, Los Alamitos, CA (1982)

Karnaugh, M.: The map method for synthesis of combinational logic circuits. Trans. AIEE, Commun. Electron. **72**, 593–599 (1953)

Karoński, M., Scheinerman, E.R., Singer-Cohen, K.: On Random Intersection Graphs: The Subgraph Problem. Comb. Probab. Comput. **8**, 131–159 (1999)

Karp, B., Kung, H.: GPSR: Greedy Perimeter Stateless Routing for Wireless Networks. In: Proc. 6th Annual Int. Conf. on Mobile Computing and Networking (MobiCom), 2000, pp 243–254

Karp, R. Raghavan, P.: From a personal communication cited in [14]

Karp, R.: A 2*k*-competitive algorithm for the circle. Manuscript (1989)

Karp, R., Pippenger, N., Sipser, M.: A Time-Randomness tradeoff. In: Proc. Conference on Probabilistic Computational Complexity, AMS, 1985, pp. 150–159

Karp, R.M.: Reducibility among combinatorial problems. In: Miller, R.E., Thatcher, J.W. (eds.) Complexity of Computer Computations, pp. 85–103. Plenum Press, New York (1972)

Karp, R.M., Lipton, R.J.: Some Connections Between Nonuniform and Uniform Complexity Classes. In: Proc. 12th Ann. ACM Symposium on Theory of Computing, 1980, pp. 302–309

Karp, R.M., Miller, R.E., Rosenberg, A.L.: Rapid identification of repeated patterns in strings, trees and arrays. In: Proc. 4th Annual ACM Symposium on Theory of Computing, pp. 125–136. ACM Press, New York (1972)

Karp, R.M., Ramachandran, V.: Parallel Algorithms for Shared-Memory Machines. In: Van Leeuwen Ed, J. (ed) Handbook of Theoretical Computer Science, vol. A, pp. (869–941). MIT Press, Massachusetts (1990)

Karp, R.M., Upfal, E., Wigderson, A.: Constructing a perfect matching is in Random NC. Combin. **6**, 35–48 (1986)

Karzanov, A.V., Timofeev, E. A.: Efficient algorithm for finding all minimal edge cuts of a nonoriented graph. Cybernetics **22**, 156–162 (1986)

Kasai, T., Lee, G., Arimura, H., Arikawa, S., Park, K.: Linear-time longest-common-prefix computation in suffix arrays and its applications. In: Proc. 12th Annual Symposium on Combinatorial Pattern Matching, vol. (2089) of LNCS. pp. 181–192. Springer, Berlin/Heidelberg (2001)

Kashyap, S., Khuller, S.: Algorithms for non-uniform size data placement on parallel disks. In: Conference on FST&TCS Conference. LNCS, vol. 2914, pp. 265–276. Springer, Heidelberg (2003)

Kashyap, S., Khuller, S., Wan, Y-C., Golubchik, L.: Fast reconfiguration of data placement in parallel disks. In: Workshop on Algorithm Engineering and Experiments (2006)

Kato, A.: Complexity of the sex-equal stable marriage problem. Jpn. J. Ind. Appl. Math. **10**, 1–19 (1993)

Katriel, I., Sanders, P., Träff, J.L.: A practical minimum spanning tree algorithm using the cycle property. In: Proc. 11th Annual European Symposium on Algorithms. LNCS, vol. 2832, pp. 679–690. Springer, Berlin (2003)

Katz, J., Koo, C.: On Expected Constant-Round Protocols for Byzantine Agreement. In: Proceedings of Advances in Cryptology–CRYPTO 2006, Santa Barbara, California, August 2006, pp. 445–462. Springer, Berlin Heidelberg New York (2006)

Katz, J., Trevisan, L.: On the efficiency of local decoding procedures for error-correcting codes. In: Proceedings of STOC'00, pp. 80–86

Katz, R.: Contemporary logic design. Benjamin/Cummings Pub. Co. (1993)

Kauffman, L.: State models and the Jones polynomial. Topology **26**, 395–407 (1987)

Kauffman, L., Lomonaco, S.: Topological Quantum Computing and the Jones Polynomial, arXiv.org:quant-ph/0605004 (2006)

Kaufman, L., Rousseeuw, P.J.: Finding Groups in Data: An Introduction to Cluster Analysis. Wiley, New York (1990)

Kaufman, T., Litsyn, S.: Almost orthogonal linear codes are locally testable. In: Proceedings of the Forty-Sixth Annual Symposium on Foundations of Computer Science, pp. 317–326. IEEE, New York (2005)

Kaufman, T., Litsyn, S., Xie, N.: Breaking the ϵ-soundness bound of the linearity test over gf(2). Electronic Colloquium on Computational Complexity, Report TR07–098, October 2007

Kaufman, T., Ron, D.: Testing polynomials over general fields. In: Proceedings of the Forty-Fifth Annual Symposium on Foundations of Computer Science, pp. 413–422. IEEE, New York (2004)

Kautz, H., Selman, B.: Ten Challenges Redux: Recent Progress in Propositional Reasoning and Search. Proceedings 9th International Conference on Principles and Practice of Constraint Programming, pp. 1–18. Kinsale, Ireland (2003)

Kavitha, T., Mehlhorn, K., Michail, D., Paluch, K.: Strongly stable matchings in time $O(nm)$ and extension to the Hospitals-Residents problem. In: Proceedings of STACS 2004: the 21st International Symposium on Theoretical Aspects of Computer Science. Lecture Notes in Computer Science, vol. 2996, pp. 222–233. Springer, Berlin (2004)

Kavvadias, D., Pantziou, G., Spirakis, P., Zaroliagis, C.: Efficient Sequential and Parallel Algorithms for the Negative Cycle Problem. In: Algorithms and Computation – ISAAC'94. Lect. Notes Comput. Sci., vol. 834, pp.270–278. Springer, Heidelberg (1994)

Kay, R., Bucheuv, G., Pileggi, L.: EWA: Efficient Wire-Sizing Algorithm. In: Proc. Intl. Symp. on Physical Design, pp. 178–185. ACM, New York (1997)

Kaye, P., Laflamme, R., Mosca, M.: An Introduction to Quantum Computation. Oxford University Press, Oxford (2007)

Kearney, P.: Phylogenetics and the quartet method. In: Jiang, T., Xu, Y., Zhang, M.Q. (eds.) Current Topics in Computational Molecular Biology. The MIT Press, Massachusetts, pp. 111–133 (2002)

Kearns, M.: Efficient noise-tolerant learning from statistical queries. J. ACM **45**(6), 983–1006 (1998)

Kearns, M., Li, M.: Learning in the presence of malicious errors. In: Proc. 20th ACM Symp. Theory of Computing, pp. 267–280, Chicago, 2–4 May 1988

Kearns, M., Li, M.: Learning in the presence of malicious errors. SIAM J. Comput. **22**, 807–837 (1993)

Kearns, M., Schapire, R., Sellie, L.: Toward efficient agnostic learning. Mach. Learn. **17**(2-3), 115–141 (1994)

Kearns, M., Singh, S.: Near-optimal reinforcement learning in polynomial time. Mach. Learn. **49**(2–3), 209–232 (2002)

Kearns, M., Valiant, L.: Cryptographic limitations on learning boolean formulae and finite automata. J. ACM **41**(1), 67–95 (1994)

Kearns, M., Vazirani, U.: An introduction to computational learning theory. MIT Press, Cambridge (1994)

Keidar, I., Rajsbaum, S.: On the cost of fault-tolerant consensus when there are no faults-a tutorial. In: Tutorial 21th ACM Symposium on Principles of Distributed Computing, July 2002

Keil, J.M., Gutwin, C.A.: Classes of graphs which approximate the complete Euclidean graph. Discrete Comput. Geom. **7**, 13–28 (1992)

Keil, J.M., Gutwin, C.A.: The Delaunay Triangulation Closely Approximates the Complete Euclidean Graph. Discret. Comput. Geom. **7**, 13–28 (1992)

Keller, O., Kopelowitz, T., Lewenstein, M.: Parameterized LCS and edit distance are NP-Complete. Manuscript

Kellerer, H., Pferschy, U.: A new fully polynomial time approximation scheme for the knapsack problem. J. Comb. Optim. **3**, 59–71 (1999)

Kellerer, H., Pferschy, U.: Improved dynamic programming in connection with an FPTAS for the knapsack problem. J. Comb. Optim. **8**, 5-11 (2004)

Kellerer, H., Pisinger, D., Pferschy U.: Knapsack Problems. Springer, Berlin (2004)

Kellerer, H., Tautenhahn, T., Woeginger, G.J.: Approximability and nonapproximability results for minimizing total flow time on a single machine. In: Proceedings of 28th Annual ACM Symposium on the Theory of Computing (STOC '96), 1996, pp. 418–426

Kellerer, H., Tautenhahn, T., Woeginger, G.J.: Approximability and Nonapproximability Results for Minimizing Total Flow Time on a Single Machine. SIAM J. Comput. **28**(4), 1155–1166 (1999)

Kelly, F.P.: Charging and rate control for elastic traffic. Eur. Trans. Telecommun. **8**, 33–37 (1997)

Kempe, J.: Discrete quantum walks hit exponentially faster. In: Proc. RANDOM (2003)

Kempe, J.: Quantum random walks – an introductory overview. Contemp. Phys. **44**(4), 302–327 (2003)

Kendon, V., Tregenna, B.: Decoherence can be useful in quantum walks. Phys. Rev. A. **67**, 42–315 (2003)

Kennings, A., Markov, I.L.: Smoothing max-terms and analytical minimization of half-perimeter wirelength. VLSI Design **14**(3), 229–237 (2002)

Kennings, A., Vorwerk, K.: Force-directed methods for generic placement. IEEE Trans. CAD **25**(10), 2076–2087 (2006)

Kent, K., Skorin-Kapov, D.: Population monotonic cost allocation on mst's. In: Proc. of the 6th International Conference on Operational Research, Croatian Operational Research Society, Zagreb, pp. 43–48 (1996)

Kern, W., Paulusma, D.: Matching Games: The Least Core and the Nucleolus. Math. Oper. Res. **28**, 294–308 (2003)

Kernighan, B.W., Lin, S.: An efficient heuristic procedure for partitioning graphs. Bell Syst. Tech. J. **49**(2), 291–307 (1970)

Kesselman, A., Lotker, Z., Mansour, Y., Patt-Shamir, B., Schieber, B., Sviridenko, M.: Buffer overflow management in QoS switches. SIAM J. Comput. **33**(3), 563–583 (2004)

Kesselman, A., Mansour, Y., van Stee, R.: Improved competitive guarantees for QoS buffering. In: Di Battista, G., Zwick, U. (eds.) Algorithms – ESA 2003, Proceedings Eleventh Annual European Symposium. Lecture Notes in Computer Science, vol. 2380, pp. 361–373. Springer, Berlin (2003)

Kettner, L., Näher, S.: Two computational geometry libraries: LEDA and CGAL. In: Goodman, J.E., O'Rourke, J. (eds.) Handbook of Discrete and Computational Geometry, Chapter 65, pp. 1435–1463, 2nd edn. Chapman & Hall/CRC, Boca Raton (2004)

Keutzer, K.: DAGON: Technology Binding and Local Optimizations by DAG Matching. In: Proc. of the 24th Design Automation Conference **28**(1), pp. 341–347. Miami Beach, June 1987

Khachiyan, L.G.: A polynomial algorithm for linear programming. Soviet Math. Doklady **20**, 191–194 (1979)

Khandekar, R., Rao, S., Vazirani, U.: Graph partitioning using single commodity flows. In: STOC '06: Proceedings of the thirty-eighth annual ACM symposium on Theory of computing, pp. 385–390. ACM Press, New York (2006)

Khanna, S., Linial, N., Safra, S.: On the hardness of approximating the chromatic number. Combinatorica **20**, 393–415 (2000)

Khardon, R., Roth, D., Servedio, R.A.: Efficiency versus convergence of boolean kernels for on-line learning algorithms. J. Artif. Intell. Res. **24**, 341–356 (2005)

Kharitonov, M.: Cryptographic hardness of distribution-specific learning. In: Proceedings of the 25th Annual Symposium on Theory of Computing, pp. 372–381. (1993)

Khot, S.: Hardness of Approximating the Shortest Vector Problem in Lattices. J. ACM **52**(5), 789–808 (2005). Preliminary version in FOCS 2004

Khot, S.: Improved inapproximability results for max clique, chromatic number and approximate graph coloring. In: Proceedings of the 42nd annual IEEE Symposium on Foundations of Computer Science (2001) pp. 600–609.

Khot, S.: On the power of unique 2-prover 1-round games. In: Proceedings of the 34th Annual Symposium on the Theory of Computing (STOC), Montreal 2002, pp. 767–775

Khot, S.: Ruling out PTAS for graph Min-Bisection, Densest Subgraph and Bipartite Clique. In: 45th Annual IEEE Symposium on Foundations of Computer Science, pp. 136–145, Georgia Inst. of Technol., Atlanta 17–19 Oct. 2004

Khot, S., Kindler, G., Mossel, E., O'Donnell, R.: Optimal inapproximability results for MAX CUT and other 2-variable CSPs? In: Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS), Rome 2004, pp. 146–154

Khot, S., Vishnoi, N.: The Unique Games Conjecture, Integrality Gap for Cut Problems and the Embeddability of Negative-Type Metrics into ℓ_1. In: Proceedings of the 46th IEEE Symposium on Foundations of Computer Science (FOCS), Pittsburgh, October 2005, pp. 53–62

Khuller, S.: Approximation algorithms for finding highly connected subgraphs. In: Hochbaum, D. (ed.) Approximation Algorithms for $\mathcal{NP}$-Hard Problems, Chapter 6, pp. 236–265. PWS Publishing Company, Boston (1996)

Khuller, S., Kim, Y., Malekian, A.: Improved algorithms for data migration. In: 9th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (2006)

Khuller, S., Kim, Y., Wan, Y.-C.: Algorithms for data migration with cloning. SIAM J. Comput. **33**(2), 448–461 (2004)

Khuller, S., Moss, A., Naor, J.: The budgeted maximum coverage problem. Inform. Process. Lett. **70**(1), 39–45 (1999)

Khuller, S., Raghavachari, B.: Improved approximation algorithms for uniform connectivity problems. J. Algorithms **21**(2), 434–450 (1996)

Khuller, S., Raghavachari, B., Young, N.: Low-degree spanning trees of small weight. SIAM J. Comput. **25**(2), 355–368 (1996)

Khuller, S., Vishkin, U.: Biconnectivity approximations and graph carvings. J. ACM **41**(2), 214–235 (1994)

Kida, T., Matsumoto, T., Shibata, Y., Takeda, M., Shinohara, A., Arikawa, S.: Collage systems: a unifying framework for compressed pattern matching. Theor. Comput. Sci. **298**(1), 253–272 (2003)

Kida, T., Takeda, M., Shinohara, A., Miyazaki, M., Arikawa, S.: Multiple pattern matching in LZW compressed text. J. Discret. Algorithms **1**(1), 133–158 (2000)

Kierstead, H.A.: The linearity of first-fit coloring of interval graphs. SIAM J. Discret. Math. **1**(4), 526–530 (1988)

Kierstead, H.A., Trotter, W.T.: An extremal problem in recursive combinatorics. Congr. Numerantium **33**, 143–153 (1981)

Kim, D.K., Kim, Y.A., Park, K.: Constructing suffix arrays for multi-dimensional matrices. In: Proceedings of the 9th Symposium on Combinatorial Pattern Matching, 1998, pp. 249–260

Kim, D.K., Kim, Y.A., Park, K.: Generalizations of suffix arrays to multi-dimensional matrices. Theor. Comput. Sci. **302**, 401–416 (2003)

Kim, D.K., Na, J.C., Kim, J.E., Park, K.: Efficient implementation of Rank and Select functions for succinct representation. In: Proc. WEA 2005. LNCS, vol. 3505, pp. 315–327 (2005)

Kim, D.K., Park, K.: Linear-time construction of two-dimensional suffix trees. In: Proceedings of the 26th International Colloquium on Automata, Languages, and Programming, 1999, pp. 463–372

Kim, D.K., Sim, J.S., Park, H., Park, K.: Constructing suffix arrays in linear time. J. Discret. Algorithms **3**, 126–142 (2005)

Kim, J.-H.: On Brook's Theorem for sparse graphs. Combin. Probab. Comput. **4**, 97–132 (1995)

Kim, J.W., Amir, A., Landau, G.M., Park, K.: Computing Similarity of Run-Length Encoded Strings with Affine Gap Penalty. In: Proc. 12th Symposium on String Processing and Information Retrieval (SPIRE'05). LNCS, vol. 3772, pp. 440–449 (2005)

Kim, S.K.: Linear-time algorithm for finding a maximum-density segment of a sequence. Inf. Process. Lett. **86**, 339–342 (2003)

Kim, Y.J., Govindan, R., Karp, B., Shenker, S.: Geographic Routing Made Practical. In: Proceedings of the Second USENIX/ACM Symposium on Networked System Design and Implementation (NSDI 2005), Boston, Massachusetts, USA, May 2005

Kim, Y.J., Govindan, R., Karp, B., Shenker, S.: Lazy cross-link removal for geographic routing. In: Embedded Networked Sensor Systems. ACM, New York (2006)

Kim, Y.J., Govindan, R., Karp, B., Shenker, S.: On the Pitfalls of Geographic Face Routing. In: Proc. of the ACM Joint Workshop on Foundations of Mobile Computing (DIALM-POMC), Cologne, Germany, September 2005

Kimbrel, T., Sinha, R.K.: A probabilistic algorithm for verifying matrix products using $O(n^2)$ time and $\log_2 n + O(1)$ random bits. Inf. Proc. Lett. **45**(2), 107–110 (1993)

Kinber, E.B., Stephan, F.: Language Learning from Texts: Mind-changes, Limited Memory, and Monotonicity. Inform. Comput. **123**(2), 224–241 (1995)

King, V.: A simpler minimum spanning tree verification algorithm. Algorithmica **18**(2), 263–270 (1997)

King, V.: Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs. In: Proc. 40th IEEE Symposium on Foundations of Computer Science (FOCS'99), pp. 81–99. IEEE Computer Society, New York, USA (1999)

King, V., Sagert, G.: A fully dynamic algorithm for maintaining the transitive closure. J. Comp. Syst. Sci. **65**(1), 150–167 (2002)

King, V., Thorup, M.: A space saving trick for dynamic transitive closure and shortest path algorithms. In: Proceedings of the 7th Annual International Conference of Computing and Cominatorics, vol. 2108/2001, pp. 269–277. Lect. Notes Comp. Sci. COCOON Springer, Heidelberg (2001)

King, V., Zhang, L., Zhou, Y.: On the complexity of distance-based evolutionary tree construction. In: Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2003), pp. 444–453 (2003)

Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by Simulated Annealing. Science **4598**, 671–680 (1983)

Kirousis, L., Stamatiou, Y., Zito, M.: The unsatisfiability threshold conjecture: the techniques behind upper bound improvements. In: A. Percus, G. Istrate, C. Moore (eds.) Computational Complexity and Statistical Physics, Santa Fe Institute Studies in the Sciences of Complexity, pp. 159–178. Oxford University Press, New York (2006)

Kirousis, L.M., Kranakis, E., Vitányi, P.M.B.: Atomic multireader register. In: Proc. Workshop Distributed Algorithms. Lect Notes Comput Sci, vol 312, pp. 278–296. Springer, Berlin (1987)

Kirousis, L.M., Spirakis, P., Tsigas, P.: Simple atomic snapshots: A linear complexity solution with unbounded time-stamps. Inf. Process. Lett. **58**, 47–53 (1996)

Kis, T., Kapolnai, R.: Approximations and auctions for scheduling batches on related machines. Operat. Res. Let. **35**(1), 61–68 (2006)

Kitaev, A.: Quantum measurements and the Abelian Stabilizer Problem. quant-ph/9511026, http://arxiv.org/abs/quant-ph/9511026 (1995) and in: Electronic Colloquium on Computational Complexity (ECCC) 3, Report TR96-003,http://eccc.hpi-web.de/eccc-reports/1995/TR96-003/ (1996)

Kitaev, A.Y.: Quantum computations: algorithms and error correction. Russ. Math. Surv. **52**(6), 1191–1249 (1997)

Kitts, B., Leblanc, B.: Optimal bidding on keyword auctions. Electronic Markets, Special issue: Innovative Auction Markets **14**(3), 186–201 (2004)

Kivinen, J., Warmuth, M.K.: Exponentiated gradient versus gradient descent for linear predictors. Inf. Comp. **132**(1), 1–64 (1997)

Kiwi, M., Magniez, F., Santha, M.: Approximate testing with error relative to input size. J. CSS **66**(2), 371–392 (2003)

Kiwi, M., Magniez, F., Santha, M.: Exact and approximate testing/correcting of algebraic functions: A survey. Theoretical Aspects Compututer Science, LNCS **2292**, 30–83 (2001)

Klasing, R., Navarra, A., Papadopoulos, A., Perennes, S.: Adaptive broadcast consumption (ABC), a new heuristic and new bounds for the minimum energy broadcast routing problem. In: Proceeding of the 3rd IFIP-TC6 international networking conference (NETWORKING), pp. 866–877 (2004)

Kleene, S.C.: Representation of events in nerve sets. In: Shannon, C.E., McCarthy, J. (eds.) Automata Studies, pp. 3–40. Princeton Univ. Press, Princeton (1956)

Klein, M., Ralya, T., Pollak, B., Obenza, R., Harbour, M.G.: A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems. Kluwer Academic Publishers, Boston (1993)

Klein, P., Agrawal, A., Ravi, R., Rao, S.: Approximation through multicommodity flow. In: Proceedings of the 31st IEEE Symposium on Foundations of Computer Science (FOCS), pp. 726–737 (1990)

Klein, P., Plotkin, S.A., Rao, S.: Excluded minors, network decomposition, and multicommodity flow. In: 25th Annual ACM Symposium on Theory of Computing, pp. 682–690, San Diego, 1993 May 16–18

Klein, P., Ravi, R.: A nearly best-possible approximation algorithm for node-weighted Steiner trees. J. Algorithms **19**(1), 104–115 (1995)

Klein, P.N.: A linear-time approximation scheme for TSP for planar weighted graphs. In: Proceedings of the 46th IEEE Symposium on Foundations of Computer Science, 2005, pp. 146–155

Klein, P.N.: A subset spanner for planar graphs, with application to subset TSP. In: Proceedings of the 38th ACM Symposium on Theory of Computing, 2006, pp. 749–756

Klein, P.N.: Multiple-source shortest paths in planar graphs. In: Proceedings, 16th ACM-SIAM Symposium on Discrete Algorithms, pp. 146–155 (2005)

Klein, P.N., Krishnan, R., Raghavachari, B., Ravi, R.: Approximation algorithms for finding low-degree subgraphs. Networks **44**(3), 203–215 (2004)

Klein, P.N., Plotkin, S.A., Stein, C., Tardos, É.: Faster approximation algorithms for the unit capacity concurrent flow problem with applications to routing and finding sparse cuts. SIAM J. Comput. **23**(3), 466–487 (1994)

Klein, R., Kutz, M.: The Density of Iterated Plane Intersection Graphs and a Gap Result for Triangulations of Finite Point Sets. In: Proc. 22nd ACM Symp. Comp. Geom. (SoCG), Sedona (AZ), 2006, pp. 264–272

Klein, S.T., Shapira, D.: Compressed pattern matching in jpeg images. In: Proceeding Prague Stringology conference, 2005, pp. 125–134

Klein, S.T., Wiseman, Y.: Parallel huffman decoding with applications to jpeg files. Comput. J. **46**(5), 487–497 (2003)

Kleinberg, J.: The small-world phenomenon: An algorithmic perspective. In: Proc. 32nd ACM Symposium on Theory of Computing (STOC 2000), 2000, pp. 163–170

Kleinberg, J., Rabani, Y., Tardos, E.: Allocating Bandwidth for Bursty Connections. SIAM J. Comput. **30**, 191–217 (2000)

Kleinberg, J., Rabani, Y., Tardos, É.: Fairness in routing and load balancing. In: Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science, pp. 568–578, October 1999

Kleinberg, J.M.: An Approximation Algorithm for the Disjoint Paths Problem in Even-Degree Planar Graphs. Proc. of IEEE FOCS, 2005, pp. 627–636

Kleinberg, J.M.: Approximation algorithms for disjoint paths problems. Ph. D. thesis, MIT, Cambridge, MA (1996)

Kleinberg, J.M.: The localization problem for mobile robots. In: Proceedings of the 35th Symposium on Foundations of Computer Science (FOCS'94), 1994, pp. 521–531

Kleinrock, L., Silvester, J.: Optimum transmission radii for packet radio networks or why six is a magic number. In: Proceedings of the IEEE National Telecommunications Conference, pp. 431–435, Birmingham, 4–6 December 1978

Kleitman, D.J., West, D.B.: Spanning trees with many leaves. SIAM J. Discret. Math. **4**, 99–106 (1991)

Klimov, G.P.: Time-sharing service systems I. Theory Probab. Appl. **19**, 532–551 (1974)

Klivans, A., O'Donnell, R., Servedio, R.: Learning intersections and thresholds of halfspaces. J. Comput. Syst. Sci. **68**(4), 808–840 (2004)

Klivans, A., Servedio, R.: Learning DNF in time $2^{\tilde{O}(n^{1/3})}$. In: Proceedings of the 33rd Annual Symposium on Theory of Computing, 2001

Klivans, A., Sherstov, A.: Cryptographic hardness results for learning intersections of halfspaces. In: Proceedings of the 47th Annual Symposium on Foundations of Computer Science, pp. 553–562, Berkeley, 22–24 October 2006

Klivans, A.R. Servedio, R.A.: Toward attribute efficient learning of decision lists and parities. J. Mach. Learn. Res. **7**(Apr), 587–602 (2006)

Klivans, A.R., Servedio, R.A.: Boosting and hard-core set construction. Mach. Learn. **51**, 217–238 (2003)

Klivans, A.R., Servedio, R.A.: Learning DNF in Time $2^{\tilde{O}(n^{1/3})}$. J. Comput. Syst. Sci. **68**, 303–318 (2004)

Kloks, T., Kratochvíl, J., Müller, H.: Computing the branchwidth of interval graphs. Discret. Appl. Math. **145**, 266–275 (2005)

Kloks, T., Kratsch, D., Spinrad, J.: On treewidth and minimum fill-in of asteroidal triple-free graphs. Theor. Comput. Sci. **175**, 309–335 (1997)

Knauer, C., Spillner, A.: A fixed-parameter algorithm for the minimum weight triangulation problem based on small graph separators. In: Proceedings of the 32nd International Workshop on Graph-Theoretic Concepts in Computer Science (WG). Lecture Notes in Computer Science, vol. 4271, pp. 49–57. Springer, New York (2006)

Knauer, C., Spillner, A.: Fixed-parameter algorithms for finding crossing-free spanning trees in geometric graphs. Tech. Rep. 06–07, Department of Computer Science, Friedrich-Schiller-Universität Jena (2006)

Kneis, J., Mölle, D., Richter, S., Rossmanith, P.: Algorithms Based on the Treewidth of Sparse Graphs. In: Proc. Workshop on Graph Theoretic Concepts in Computer Science. LNCS, vol. 3787, pp. 385–396. Springer, Berlin (2005)

Knill, E.: Quantum computing with realistically noisy devices. Nature **434**, 39–44 (2005)

Knill, E., Laflamme, R.: Theory of quantum error-correcting codes. Phys. Rev. A **55**, 900–911 (1997)

Knill, E., Laflamme, R., Martinez, R., Negrevergne, C.: Benchmarking quantum computers: the five-qubit error correcting code. Phys. Rev. Lett. **86**, 5811–5814 (2001)

Knödel, W.: A bin packing algorithm with complexity $O(nlogn)$ in the stochastic limit. In: Proc. 10th Symp. on Mathematical Foundations of Computer Science. LNCS, vol. 118, pp. 369–378. Springer, Berlin (1981)

Knopp, S., Sanders, P., Schultes, D., Schulz, F., Wagner, D.: Computing many-to-many shortest paths using highway hierarchies. In: Proceedings 9th Workshop on Algorithm Engineering and Experiments (ALENEX), 2007

Knuth, D.: Marriage Stables et leurs relations avec d'autres problèmes Combinatories. Les Presses de l'Université de Montréal (1976)

Knuth, D.: The Art of Computer Programming, vol. 2 : Seminumerical Algorithms, 2nd edn. Addison-Wesley Publishing Company, Reading (1981)

Knuth, D., Plass, M.: Breaking paragraphs into lines. Software-Practice Exp. **11**, 1119–1184 (1981)

Knuth, D.E.: Mariages Stables. Les Presses de L'Université de Montréal, Montréal (1976)

Knuth, D.E.: Optimum binary search trees. Acta Informatica **1**, 14–25 (1971)

Knuth, D.E.: Sorting and Searching. The Art of Computer Programming, vol. 3, 2nd edn. Addison-Wesley, Reading (1998)

Knuth, D.E., Morris, J.H. Jr., Pratt, V.R.: Fast pattern matching in strings. SIAM J. Comput. **6**(1), 323–350 (1977)

Ko, P., Aluru, S.: Optimal self-adjusting trees for dynamic string data in secondary storage. In: Symposium on String Processing and Information Retrieval (SPIRE). LNCS, vol. 4726, pp. 184-194. Springer, Berlin (2007)

Ko, P., Aluru, S.: Space efficient linear time construction of suffix arrays. J. Discret. Algorithms **3**, 143–156 (2005)

Köbler, J., Lindner, W.: Oracles in s^{P}_{2} are sufficient for exact learning. Int. J. Found. Comput. Sci. **11**(4), 615–632 (2000)

Köbler, J., Schöning, U., Torán, J.: The Graph Isomorphism Problem: its structural complexity. Birkhäuser, Boston (1993)

Köbler, J., Watanabe, O.: New Collapse Consequences of NP Having Small Circuits. SIAM J. Comput. **28**, 311–324 (1998)

Kodama, C., Fujiyoshi, K.: Selected Sequence-Pair: An efficient decodable packing representation in linear time using Sequence-Pair. In: Proc. ASP-DAC 2003, pp. 331–337

Koenig, S., Mudgal, A., Tovey, C.: A near-tight approximation lower bound and algorithm for the kidnapped robot problem. In:

Proceedings of the 17th ACM-SIAM Symposium on Discrete Algorithms (SODA'06), 2006, pp. 133–142.

Koetter, R., Vardy, A.: Algebraic soft-decision decoding of Reed–Solomon codes. IEEE Trans. Inf. Theory. **49**(11), 2809–2825 (2003)

Koetter, R., Vontobel, P.: Graph covers and iterative decoding of finite-length codes. In: Proc. 3rd International Symposium on Turbo Codes and Related Topics, pp. 75–82, September 2003. Brest, France (2003)

Köhler, E., Möhring, R., Schilling, H.: Fast point-to-point shortest path computations with arc-flags. In: 9th DIMACS Implementation Challenge Workshop: Shortest Paths, DIMACS Center, Piscataway, NJ, 13–14 Nov 2006

Kojevnikov, A., Kulikov, A.S.: A New Approach to Proving Upper Bounds for Max 2-SAT. In: Proc. of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 11–17 (2006)

Kojima, F., Unver, Ü.: Random paths to pairwise stability in many-to-many matching problems: A study on market equilibration. Intern. J. Game Theor. (2006)

Kolchin, V.F., Sevastyanov, B.A., Chistyakov, V.P.: Random Allocations. Wiley, New York (1978)

Kolliopoulos, S.G.: Edge Disjoint Paths and Unsplittable Flow. In: Handbook on Approximation Algorithms and Metaheuristics, Chapman & Hall/CRC Press Computer & Science Series, vol 13. Chapman Hall/CRC Press, May 2007

Kolliopoulos, S.G., Stein, C.: Approximating Disjoint-Path Problems Using Greedy Algorithms and Packing Integer Programs. Math. Program. A **99**, 63–87 (2004). Preliminary version in Proc. of IPCO 1998

Kolliopoulos, S.G., Stein, C.: Finding Real-Valued Single-Source Shortest Paths in $o(n^3)$ Expected Time. J. Algorithms **28**, pp. 125–141 (1998)

Kolliopoulos, S.G., Young, N.E.: Tight approximation results for general covering integer programs. In: Proceedings of the forty-second annual IEEE Symposium on Foundations of Computer Science, pp. 522–528 (2001)

Kolpakov, R., Bana, G., Kucherov, G.: `mreps`: efficient and flexible detection of tandem repeats in DNA. Nucl. Acids Res. **31**(13), 3672–3678 (2003)

Kolpakov, R., Kucherov, G.: Finding approximate repetitions under Hamming distance. Theoret. Comput. Sci. **33**(1), 135–156, (2003)

Kolpakov, R., Kucherov, G.: Finding maximal repetitions in a word in linear time. In: Proceedings of the 40th Symposium on Foundations of Computer Science, pp. 596–604. IEEE Computer Society Press, Los Alamitos (1999)

Kolpakov, R., Kucherov, G.: Identification of periodic structures in words. In: Berstel, J., Perrin, D. (eds.) Applied combinatorics on words. Encyclopedia of Mathematics and its Applications. Lothaire books, vol. 104, pp. 430–477. Cambridge University Press (2005)

Komlós, J.: Linear verification for spanning trees. Combinatorica **5**(1), 57–65 (1985)

Komlós, J., Szemerédi, E.: Limit Distributions for the existence of Hamilton cycles in a random graph. Discret. Math. **43**, 55–63 (1983)

Könemann, J., Leonardi, S., Schäfer, G.: A group-strategyproof mechanism for Steiner forests. In: Proc. of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 612–619. Society for Industrial and Applied Mathematics, Philadelphia (2005)

Könemann, J., Ravi, R.: A matter of degree: Improved approximation algorithms for degree-bounded minimum spanning trees. SIAM J. Comput. **31**(6), 1783–1793 (2002)

Könemann, J., Ravi, R.: Primal-dual meets local search: Approximating MSTs with nonuniform degree bounds. SIAM J. Comput. **34**(3), 763–773 (2005)

Kontogiannis, S., Panagopoulou, P.N., Spirakis, P.G.: Polynomial algorithms for approximating Nash equilibria of bimatrix games. In: Proceedings of the 2nd Workshop on Internet and Network Economics (WINE'06), pp. 286–296. Patras, 15–17 December 2006

Kontogiannis, S., Spirakis, P.G.: Efficient Algorithms for Constant Well Supported Approximate Equilibria in Bimatrix Games. In: Proceedings of the 34th International Colloquium on Automata, Languages and Programming (ICALP'07, Track A: Algorithms and Complexity), Wroclaw, 9–13 July 2007

Kopelowitz, A.: Computation of the Kernels of Simple Games and the Nucleolus of *n*-person Games. RM-31, Math. Dept., The Hebre University of Jerusalem (1967)

Korach, E., Moran, S., Zaks, S.: The optimality of distributive constructions of minimum weight and degree restricted spanning trees in a complete network of processors. In: Proc. 4th Symp. on Principles of Distributed Computing (PODC), pp. 277–286. ACM, USA (1985)

Korach, E., Moran, S., Zaks, S.: Tight upper and lower bounds for some distributed algorithms for a complete network of processors. In: Proc. 3rd Symp. on Principles of Distributed Computing (PODC), pp. 199–207. ACM, USA (1984)

Korilis, Y.A., Lazar, A.A., Orda, A.: Achieving network optima using stackelberg routing strategies. IEEE/ACM Trans. Netw. **5**(1), 161–173 (1997)

Korshunov, A.D.: Solution of a problem of P. Erdös and A. Rényi on Hamilton Cycles in non-oriented graphs. Metody Diskr. Anal. Teoriy Upr. Syst. Sb. Trubov Novosibirrsk **31**, 17–56 (1977)

Korte, B., Schrader, R.: On the existence of fast approximation schemes. Nonlinear Program. **4**, 415–437 (1980)

Korupolu, M.R., Plaxton, C.G., Rajaraman, R.: Analysis of a local search heuristic for facility location problems. In: SODA '98: Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms, pp. 1–10. San Francisco, USA; 25–26 January 1998

Kosaraju, R., Manzini, G.: Compression of low entropy strings with Lempel–Ziv algorithms. SIAM J. Comput. **29**, 893–911 (1999)

Kosaraju, S.R.: Efficient tree pattern matching. In: Proc. 20th IEEE Foundations of Computer Science (FOCS), pp. 178–183. Triangle Park, USA (1989)

Kosaraju, S.R.: Faster algorithms for the construction of parameterized suffix trees. In: Proc. 36th Annual Symposium on Foundations of Computer Science (FOCS), 1995, pp. 631–637

Koster, A.M.C.A., Bodlaender, H.L., van Hoesel, S.P.M.: Treewidth: Computational experiments. In: Broersma, H., Faigle, U., Hurink, J., Pickl, S. (eds.) Electronic Notes in Discrete Mathematics, vol. 8, pp. 54–57. Elsevier, Amsterdam (2001)

Kothari, A., Parkes, D., Suri, S.: Approximately-strategyproof and tractable multi-unit auctions. Decis. Support Syst. **39**, 105–121 (2005)

Kouider, M., Vestergaard, P.D.: Generalized connected domination in graphs. Discret. Math. Theor. Comput. Sci. (DMTCS) **8**, 57–64 (2006)

Koutsoupias, E.: Weak adversaries for the k-server problem. In: Proc. 40th Symp. Foundations of Computer Science (FOCS), IEEE, pp. 444–449 (1999)

Koutsoupias, E., Mavronicolas, M., Spirakis, P.: Approximate equilibria and ball fusion. Theor. Comput. Syst. **36**(6), 683–693 (2003)

Koutsoupias, E., Papadimitriou, C.: On the *k*-server conjecture. In: Proc. 26th Symp. Theory of Computing (STOC), pp. 507–511. ACM (1994)

Koutsoupias, E., Papadimitriou, C.: The 2-evader problem. Inf. Proc. Lett. **57**, 249–252 (1996)

Koutsoupias, E., Papadimitriou, C.: Worst-case equilibria. In: 16th Symposium on Theoretical Aspects in Computer Science, Trier, Germany. LNCS, vol. 1563, pp. 404–413. Springer (1999)

Koutsoupias, E., Papadimitriou, C.H.: Beyond competitive analysis. In: Proceeding 35th Annual Symposium on Foundations of Computer Science, pp. 394–400, Santa Fe, NM (1994)

Koutsoupias, E., Papadimitriou, C.H.: Beyond competitive analysis. SIAM J. Comput. **30**(1), 300–317 (2000)

Koutsoupias, E., Papadimitriou, C.H.: On the greedy algorithm for satisfiability. Inform. Process. Lett. **43**(1), 53–55 (1992)

Koutsoupias, E., Papadimitriou, C.H.: On the k-server conjecture. J. ACM **42**(5), 971–983 (1995)

Koutsoupias, E., Papadimitriou, C.H.: Worst-case equilibria. In: Proc. of the 16th Annual Symposium on Theoretical Aspects of Computer Science (STACS), pp. 404–413. Springer, Trier (1999)

Koutsoupias, E., Taylor, D.S.: The CNN problem and other k-server variants. Theor. Comput. Sci. **324**, 347–359 (2004)

Kovács, A.: Fast Algorithms for Two Scheduling Problems. Ph. D. thesis, Universität des Saarlandes (2007)

Kovács, A.: Fast monotone 3-approximation algorithm for scheduling related machines. In: Proc. 13th Annual European Symposium on Algorithms (ESA), 2005, pp. 616–627

Kowalski, D.R., Pelc, A.: Broadcasting in undirected ad hoc radio networks. Distrib. Comput. **18**(1), 43–57 (2005)

Kowalski, D.R., Pelc, A.: Deterministic broadcasting time in radio networks of unknown topology. In: FOCS '02: Proceedings of the 43rd Symposium on Foundations of Computer Science, Washington, DC, USA, pp. 63–72. IEEE Computer Society (2002)

Kozlov, M.K., Tarasov, S.P., Khachiyan, L.G.: Polynomial solvability of convex quadratic programming. Sov. Math. Dokl. **20**, 1108–1111 (1979)

Kozma, G., Lotker, Z., Sharir, M., Stupp, G.: Geometrically aware communication in random wireless networks. In: Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing, 25–28 July 2004, pp. 310–319

Kranakis, E., Krizanc, D., Markou, E.: Mobile Agent Rendezvous in a Synchronous Torus. In: Proceedings of LATIN 2006, 7th Latin American Symposium. Valdivia, March 20–24 2006. Correa, J., Hevia, A., Kiwi, M. SVLNCS **3887**, 653–664 (2006)

Kranakis, E., Krizanc, D., Pelc, A.: Fault-tolerant broadcasting in radio networks. J. Algorithms **39**, 47–67 (2001)

Kranakis, E., Krizanc, D., Santoro, N., Sawchuk, C.: Mobile Agent Rendezvous Search Problem in the Ring. In: Proc. International Conference on Distributed Computing Systems (ICDCS), pp. 592–599. Providence, Rhode Island 19–22 May 2003

Kranakis, E., Singh, H., Urrutia, J.: Compass Routing on Geometric Networks. In: Proc. 11th Canadian Conference on Computational Geometry, Vancouver, August 1999, pp 51–54

Krarup, J., Pruzan, P.M.: Ingredients of locational analysis. In: Mirchandani, P., Francis, R. (eds.) Discrete Location Theory, pp. 1–54. Wiley, New York (1990)

Krarup, J., Pruzan, P.M.: The simple plant location problem: Survey and synthesis. Eur. J. Oper. Res. **12**, 38–81 (1983)

Kratsch, D.: Algorithms. In: Haynes, T., Hedetniemi, S., Slater, P. (eds.) Domination in Graphs: Advanced Topics, pp. 191–231. Marcel Dekker, New York (1998)

Kratsch, D., Spinrad, J.: Minimal fill in $O(n^{2.69})$ time. Discret. Math. **306**(3), 366–371 (2006)

Krauthgamer, R., Lee, J.R., Mendel, M., Naor, A.: Measured descent: A new embedding method for finite metrics. Geom. Funct. Anal. **15**(4), 839–858 (2005)

Krauthgamer, R., Linial, N., Magen, A.: Metric embeddings–beyond one-dimensional distortion. Discrete Comput. Geom. **31**(3), 339–356 (2004)

Krauthgamer, R., Rabani, Y.: Improved lower bounds for embeddings into l_1. In: SODA '06: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm, pp. 1010–1017. ACM Press, New York (2006)

Kribs, D., Laflamme, R., Poulin, D.: Unified and generalized approach to quantum error correction. Phys. Rev. Lett. **94**(4), 180501 (2005)

Krishnan, P., Vitter, J.: Optimal prediction for prefetching in the worst case. SIAM J. Comput. **27**, 1617–1636 (1998)

Krithivasan, K., Sitalakshmi, R.: Efficient Two-Dimensional Pattern Matching in The Presence of Errors. Inf. Sci. **43**, 169–184 (1987)

Krivelevich, M., Vilenchik, D.: Solving random satisfiable 3CNF formulas in expected polynomial time. In: SODA '06: Proceedings of the 17th annual ACM-SIAM symposium on Discrete algorith. ACM, Miami, Florida (2006)

Krysta, P.: Greedy approximation via duality for packing, combinatorial auctions and routing. In: Proc. 30th Int. Conference on Mathematical Foundations of Comput. Sci. (MFCS). Lecture Notes in Computer Science, vol. 3618, pp. 615–627 (2005)

Krysta, P., Kumar, V.S.A.: Approximation algorithms for minimum size 2-connectivity problems. In: Ferreira, A., Reichel, H. (eds.) STACS. Lecture Notes in Computer Science, vol. 2010, pp. 431–442. Springer, Berlin (2001)

Krznaric, D., Levcopoulos, C., Nilsson, B.J.: Minimum Spanning Trees in d Dimensions. Nord. J. Comput. **6**(4), 446–461 (1999)

Kubicka, E., Kubicki, G., McMorris, F.R.: An algorithm to find agreement subtrees. J. Classific. **12**, 91–100 (1995)

Kuehlmann, A., Krohm, F.: Equivalence Checking Using Cuts and Heaps. In: ACM Design Automation Conference (1997)

Kuehn, A.A., Hamburger, M.J.: A heuristic program for locating warehouses. Management Sci. **9**(4), 643–666 (1963)

Kuhn, F., Moscibroda, T., Nieberg, T., Wattenhofer, R.: Fast deterministic distributed maximal independent set computation on growth-bounded graphs. In: Proc. of th 19th Int. Conference on Distributed Computing (DISC), pp. 273–287 (2005)

Kuhn, F., Moscibroda, T., Nieberg, T., Wattenhofer, R.: Local approximation schemes for ad hoc and sensor networks. In: Proc. of the 3rd Joint Workshop on Foundations of Mobile Computing (DIALM-POMC), pp. 97–103 (2005)

Kuhn, F., Moscibroda, T., Wattenhofer, R.: Initializing Newly Deployed Ad Hoc and Sensor Networks. In: Proc. of the 10th Annual International Conference on Mobile Computing and Networking (MOBICOM), pp. 260–274 (2004)

Kuhn, F., Moscibroda, T., Wattenhofer, R.: On the locality of bounded growth. In: Proc. of the 24th ACM Symposium on Principles of Distributed Computing (PODC), pp. 60–68 (2005)

Kuhn, F., Moscibroda, T., Wattenhofer, R.: The price of being near-sighted. In: Proc. of the 17th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 980–989 (2006)

Kuhn, F., Moscibroda, T., Wattenhofer, R.: What cannot be computed locally! In: Proc. of the 23rd ACM Symp. on Principles of Distributed Computing (PODC), pp. 300–309 (2004)

Kuhn, F., Wattenhofer, R.: Constant-time distributed dominating set approximation. Distrib. Comput. **17**(4), 303–310 (2005)

Kuhn, F., Wattenhofer, R.: Constant-Time Distributed Dominating Set Approximation. In: PODC, Boston, Massachusetts, USA, July 2003

Kuhn, F., Wattenhofer, R., Zhang, Y., Zollinger, A.: Geometric ad-hoc routing: of theory and practice. In: Principles of Distributed Computing. ACM, New York (2003)

Kuhn, F., Wattenhofer, R., Zhang, Y., Zollinger, A.: Geometric ad-hoc routing: Of theory and practice. In: Proceedings of the Twenty-Second ACM Symposium on the Principles of Distributed Computing, Boston, Massachusetts, July 2003, pp. 63–72

Kuhn, F., Wattenhofer, R., Zollinger, A.: Ad-Hoc Networks Beyond Unit Disk Graphs. In: 1st ACM Joint Workshop on Foundations of Mobile Computing (DIALM-POMC), San Diego, California, USA, September 2003

Kuhn, F., Wattenhofer, R., Zollinger, A.: Asymptotically Optimal Geometric Mobile Ad-Hoc Routing. In: Proc. 6th Int. Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (Dial-M), pp 24–33. ACM Press, New York (2002)

Kuhn, F., Wattenhofer, R., Zollinger, A.: Worst-case optimal and average-case efficient geometric ad-hoc routing. In: Proceedings of the Fourth ACM International Symposium on Mobile Ad Hoc Networking and Computing, Annapolis, Maryland, June 2003, pp. 267–278

Kuhn, H.: The Hungarian method for the assignment problem. Naval Res. Logist. Quart. **2**, 83–97 (1955)

Kuhner, M., Felsenstein, J.: A simulation comparison of phylogeny algorithms under equal and unequal evolutionary rates. Mol. Biol. Evol. **11**(3), 459–468 (1994)

Kulikov, A.: Automated Generation of Simplification Rules for SAT and MAXSAT. Proceedings of the Eighth International Conference on Theory and Applications of Satisfiability Testing (SAT 2005). Lecture Notes in Computer Science, vol. 3569, pp. 430–436. Springer, Berlin (2005)

Kulla, F., Sanders, P.: Scalable parallel suffix array construction. In: Proc. 13th European PVM/MPI User's Group Meeting. LNCS, vol. 4192, pp. 22–29. Springer, Berlin/Heidelberg (2006)

Kullmann, O.: New methods for 3-SAT decision and worst-case analysis. Theor. Comp. Sci. **223**(1–2), 1–72 (1999)

Kullmann, O., Luckhardt, H.: Algorithms for SAT/TAUT decision based on various measures, preprint, 71 pages, http://cs-svr1.swan.ac.uk/csoliver/papers.html (1998)

Kumar, S., Lai, T.H., Balogh, J.: On *k*-coverage in a mostly sleeping sensor network. In: Proceedings of the 10th Annual International Conference on Mobile Computing and Networking (MobiCom'04), 26 Sept–1 Oct 2004

Kumar, V.S.A., Marathe, M.V.: Improved results for stackelberg scheduling strategies. In: 29th International Colloquium, Automata, Languages and Programming. LNCS, pp. 776–787. Springer (2002)

Kumar, V.S.A., Marathe, M.V., Parthasarathy, S., Srinivasan, A.: Algorithmic aspects of capacity in wireless networks. In: Proc. ACM SIGMETRICS 2005, pp. 133–144

Kumar, V.S.A., Marathe, M.V., Parthasarathy, S., Srinivasan, A.: End-to-end packet-scheduling in wireless ad-hoc networks. In: Proc. ACM-SIAM symposium on Discrete algorithms 2004, pp. 1021–1030

Kumar, V.S.A., Marathe, M.V., Parthasarathy, S., Srinivasan, A.: Scheduling on unrelated machines under tree-like precedence constraints. In: APPROX-RANDOM, pp. 146–157 (2005)

Kuo, T.-W., Mok, A.K.: Load adjustment in adaptive real-time systems. In: Proceedings of the IEEE Real-Time Systems Symposium, pp. 160–171. San Antonio, December 1991

Kuperberg, K., Kuperberg, W., Matousek, J., Valtr, P.: Almost Tiling the Plane with Ellipses. Discrete Comput. Geom. **22**(3), 367–375 (1999)

Kurose, J.F., Simha, R.: A Microeconomic Approach to Optimal Resource Allocation in Distributed Computer Systems. IEEE Trans. Comput. **38**(5), 705–717 (1989)

Kurtz, S.: Reducing the space requirements of suffix trees. Softw. Pract. Exp. **29**, 1149–1171 (1999)

Kushilevitz, E.: A simple algorithm for learning $O(\log n)$-term DNF. In: Proc. of 9th Annu. ACM Conf. on Comput. Learning Theory, pp 266–269, ACM Press, New York (1996). Journal version: Inform. Process. Lett. **61**(6), 289–292 (1997)

Kushilevitz, E., Mansour, Y.: An $\Omega(d \log(n/d))$ lower bound for broadcast in radio networks. In: PODC, 1993, pp. 65–74

Kushilevitz, E., Mansour, Y.: Learning decision trees using the Fourier spectrum. SIAM J. Comput. **22**(6), 1331–1348 (1993)

Kusner, R.B., Sullivan, J.M.: On Distortion and Thickness of Knots. In: Whittington, S.G. et al. (eds.) Topology and Geometry in Polymer Science. IMA Volumes in Math. and its Applications, vol. 103, pp. 67–78. Springer, New York (1998)

Kutin, S.: Quantum lower bound for the collision problem with small range. Theor. Comput. **1**, 29–36 (2005)

Kutten, S., Patt-Shamir, B.: Time-Adaptive Self Stabilization. In: Proceedings of the 16th Annual ACM Symposium on Principles of Distributed Computing, pp. 149–158, Santa Barbara, August 1997

Kutzelnigg, R.: Bipartite Random Graphs and Cuckoo Hashing. In: Proc. Fourth Colloquium on Mathematics and Computer Science, Nancy, France, 18–22 September 2006

Kutzschebauch, T., Stok, L.: Congestion Aware Layout Driven Logic Synthesis. In: Proc. of the IEEE/ACM International Conference on Computer-Aided Design, 2001, pp. 216–223

Kwon, W., Kim, T.: Optimal Voltage Allocation Techniques for Dynamically Variable Voltage Processors. ACM Trans. Embed. Comput. Syst. **4**(1), 211–230. New York, NY, USA (2005)

Kyasanur, P., Vaidya, N.: Capacity of multi-channel wireless networks: Impact of number of channels and interfaces. In: Proc. ACM MOBICOM, pp. 43–57. 2005

La Poutré, J.A.: Maintenance of 2- and 3-edge-connected components of graphs II. SIAM J. Comput. **29**(5), 1521–1549 (2000)

La Poutré, J.A.: Maintenance of triconnected components of graphs. In: Proc. 19th Int. Colloquium on Automata, Languages and Programming. Lecture Notes in Computer Science, vol. 623, pp. 354–365. Springer, Berlin (1992)

La Poutré, J.A., van Leeuwen, J., Overmars, M.H.: Maintenance of 2- and 3-connected components of graphs, part I: 2- and 3-edge-connected components. Discret. Math. **114**, 329–359 (1993)

La Poutré, J.A., Westbrook, J.: Dynamic two-connectivity with backtracking. In: Proc. 5th ACM-SIAM Symp. Discrete Algorithms, 1994, pp. 204–212

Ladner, R.E., Fix, J.D., LaMarca, A.: Cache performance analysis of traversals and random accesses. In: Proc. of 10th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 1999), pp. 613–622 Society for Industrial and Applied Mathematics, Philadelphia (1999)

Ladner, R.E., Fortna, R., B.-Nguyen, H.: A comparison of cache aware and cache oblivious static search trees using program instrumentation. In: Experimental Algorithmics. LNCS, vol. 2547, pp. 78–92. Springer, Berlin (2000)

Laffont, J.-J., Robert, J.: Optimal auction with financially constrained buyers. Econ. Lett. **52**, 181–186 (1996)

Lagergren, J.: Combining polynomial running time and fast convergence for the disk-covering method. J. Comput. Syst. Sci. **65**(3), 481–493 (2002)

Laird, P.: Learning from good and bad data. Kluwer Academic Publishers (1988)

Lake, J.A.: Reconstructing evolutionary trees from DNA and protein sequences: paralinear distances. Proc. Natl. Acad. Sci. USA **91**, 1455–1459 (1994)

Lakshmanan, K.B., Thulasiraman, K., Comeau, M.A.: An efficient distributed protocol for finding shortest paths in networks with negative cycles. IEEE Trans. Softw. Eng. **15**, 639–644 (1989)

Lam, T.W., Sung, W.K., Wong, S.S.: Improved approximate string matching using compressed suffix data structures. In: Proceedings of International Symposium on Algorithms and Computation, 2005, pp. 339–348

Lamport, L.: A fast mutual exclusion algorithm. ACM Trans. Comput. Syst. **5**(1), 1–11 (1987)

Lamport, L.: A new solution of Dijkstra's concurrent programming problem. Commun. ACM **17**(8), 453–455 (1974)

Lamport, L.: How to make a multiprocessor computer that correctly executes multiprocess programs. IEEE Trans. Comput. **C-28**(9), 690 (1979)

Lamport, L.: On interprocess communication. Part I: Basic formalism. Distrib. Comput. **1**, 77–85 (1986)

Lamport, L.: On interprocess communication—Part I: Basic formalism, Part II: Algorithms. Distrib. Comput. **1**(2), 77–101 (1986)

Lamport, L.: On interprocess communication, Part II: Algorithms. Distrib. Comput. **1**(2), 86–101 (1986)

Lamport, L.: The implementation of reliable distributed multiprocess systems. Comput. Netw. **2**, 95–114 (1978)

Lamport, L.: The mutual exclusion problem. Part I: A theory of interprocess communication. J. ACM **33**(2), 313–326 (1986)

Lamport, L.: The Mutual Exclusion Problem: Part II-Statement and Solutions. J. ACM **33**(2), 327–348 (1986)

Lamport, L.: The Part-Time parliament. ACM Trans. Comput. Syst. **16**(2), 133–169 (1998)

Lamport, L.: Time, clocks, and the ordering of events in a distributed system. Commun. ACM **21**(7), 558–565 (1978)

Lamport, L., Shostak, R., Pease, M.: The Byzantine generals problem. ACM Trans. Program. Lang. Syst. **4**(3), 382–401 (1982)

Lancia, G.: The phasing of heterozygous traits: Algorithms and complexity. Comput. Math. Appl. **55**(5), 960–969 (2008)

Lanckriet, G.R.G., Cristianini, N., Bartlett, P., El Ghaoui, L., Jordan, M.I.: Learning the Kernel Matrix with Semidefinite Programming. J. Mach. Learn. Res. **5**, 27–72 (2004)

Lanctot, J.K.: Li, M., Ma, B., Wang, S., Zhang, L.: Distinguishing String Search Problems. Inf. Comput. **185**, 41–55 (2003)

Lanctot, K., Li, M., Ma, B., Wang, S., Zhang, L.: Distinguishing string selection problems. In: Proc. 10th ACM-SIAM Symp. on Discrete Algorithms, pp. 633–642. (1999)

Landau, G., Vishkin, U.: Fast parallel and serial approximate string matching. J. Algorithms **10**, 157–169 (1989)

Landau, G.M., Myers, E.W., Schmidt, J.P.: Incremental string comparison. SIAM J. Comput. **27**(2), 557–582 (1998)

Landau, G.M., Schmidt, J.P., Sokol, D.: An algorithm for approximate tandem repeats. J. Comput. Biol. **8**, 1–18 (2001)

Landau, G.M., Vishkin, U.: Fast string matching with *k* differences. J. Comput. Syst. Sci. **37**(1), 63–78 (1988)

Landau, G.M., Vishkin, U.: Pattern matching in a digitized image. Algorithmica **12**(3/4), 375–408 (1994)

Lang, K., Rao, S.: Finding near-optimal cuts: an empirical evaluation. In: SODA '93: Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms, pp. 212–221. Society for Industrial and Applied Mathematics, Philadelphia (1993)

Lange, S., Grieser, G., Zeugmann, T.: Inductive inference of approximations for recursive concepts. Theor. Comput. Sci. **348**(1), 15–40 (2005)

Langville, A.N., Meyer, C.D.: Deeper Inside PageRank. Internet Math. **1**(3), 335–380 (2004)

Larsson, N.J., Sadakane, K.: Faster suffix sorting. Theor. Comput. Sci. **387**, 258–272 (2006)

Lauritzen, S.J., Spiegelhalter, D.J.: Local computations with probabilities on graphical structures and their application to expert systems. J. Royal Stat. Soc. Ser. B (Methodological) **50**, 157–224 (1988)

Lauther, U.: A Min-Cut Placement Algorithm for General Cell Assemblies Based on a Graph Representation. J. Digital Syst. **4**, 21–34 (1980)

Lauther, U.: An experimental evaluation of point-to-point shortest path calculation on roadnetworks with precalculated edgeflags. In: 9th DIMACS Implementation Challenge Workshop: Shortest Paths, DIMACS Center, Piscataway, NJ, 13–14 Nov 2006

Lavi, R., Mu'alem, A., Nisan, N.: Towards a characterization of truthful combinatorial auctions. In: Proc. of the 44rd Annual Symposium on Foundations of Computer Science (FOCS'03), 2003

Lavi, R., Nisan, N.: Competitive analysis of incentive compatible online auctions. Theor. Comput. Sci. **310**, 159–180 (2004)

Lavi, R., Nisan, N.: Online ascending auctions for gradually expiring items. In: Proc. of the 16th Symposium on Discrete Algorithms (SODA), 2005

Lavi, R., Swamy, C.: Truthful and near-optimal mechanism design via linear programming. In: Proc. 46th Annual Symposium on Foundations of Computer Science (FOCS), 2005, pp. 595–604

Lavi, R., Swamy, C.: Truthful mechanism design for multi-dimensional scheduling via cycle monotonicity (2007). Working paper

Lawler, E.L.: Combinatorial Optimization: Networks and Matroids. Holt, Rinehart and Winston, New York (1976)

Lawler, E.L.: Fast approximation algorithms for knapsack problems. Math. Oper. Res. **4**, 339–356 (1979)

Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., Shmoys, D.B. (eds.): The Traveling Salesman Problem. A Guided Tour of Combinatorial Optimization. Wiley, Chichester (1985)

Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., Shmoys, D.B.: Sequencing and Scheduling: Algorithms and Complexity. In: Graves, S.C., Rinnooy Kan, A.H.G., Zipkin, P.H. (eds.) Logistics of Production and Inventory. Handbooks in Operations Research and Management Science, vol. 4, pp. 445–522. North–Holland, Amsterdam (1993)

Lawler, E.L., Levitt, K.N., Turner, J.: Module clustering to minimize delay in digital networks. IEEE Trans. Comput. **C-18**, 47–57 (1966)

Lawrence, C., Reilly, A.: An expectation maximization (EM) algorithm for the identification and characterization of common sites in unaligned biopolymer sequences. Proteins **7**, 41–51 (1990)

Le Gall, F.: Exponential separation of quantum and classical online space complexity. In: Proc. ACM Symp. on Parallel Algorithms and Architectures (SPAA), Cambride, 30 July–1 August (2006)

LeCun, Y., Jackel, L.D., Bottou, L., Brunot, A., Cortes, C., Denker, J.S., Drucker, H., Guyon, I., Muller, U.A., Sackinger, E., Simard, P., Vapnik, V.: Comparison of learning algorithms for handwritten digit recognition. In: Fogelman-Soulie F., Gallinari P. (eds.), Proceedings International Conference on Artificial Neural Networks (ICANN) **2**, 5360. EC2 (1995)

Lee, A.W.: Diamonds are a plane graph's best friend. Master's thesis, School of Computer Science, Carleton University, Ottawa (2004)

Lee, C.-M., Hung, L.-J., Chang, M.-S., Tang, C.-Y.: An improved algorithm for the maximum agreement subtree problem. BIBE, p. 533 (2004)

Lee, C.C., Lee, D.T.: A simple on-line packing algorithm. J. ACM **32**, 562–572 (1985)

Lee, C.C., Lee, D.T.: Robust on-line bin packing algorithms. Tech. Rep. Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL (1987)

Lee, J.R.: Volume distortion for subsets of Euclidean spaces. In: Proceedings of the 22nd Annual Symposium on Computational Geometry, ACM, Sedona, AZ 2006, pp. 207–216.

Lee, S., Bhattacharjee, B., Banerjee, S.: Efficient geographic routing in multihop wireless networks. In MobiHoc '05: Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing, pp. 230–241. ACM, New York (2005)

Lehman, E., Watanabe, Y., Grodstein, J., Harkness, H.: Logic Decomposition during Technology Mapping. IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. **16**(8), 813–834, (1997)

Lehmann, B., Lehmann, D., Nisan, N.: Combinatorial auctions with decreasing marginal utilities. Games Econom. Behav. **55**(2), 270–296 (2006)

Lehmann, D., O'Callaghan, L., Shoham, Y.: Truth revelation in approximately efficient combinatorial auctions. J. ACM **49**(5), 577–602 (2002)

Lehmann, D.J., O'Callaghan, L.I., Shoham, Y.: Truth revelation in approximately efficient combinatorial auctions. In: Proc. 1st ACM Conference on Electronic Commerce (EC), pp. 96–102 (1999)

Lehoczky, J., Sha, L., Ding, Y.: The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In: Proceedings of the Real-Time Systems Symposium – 1989, Santa Monica, December 1989. IEEE Computer Society Press, pp. 166–171

Leighton, F.T.: Introduction to Parallel Algorithms and Architectures: Arrays – Trees – Hypercubes. Morgan Kaufmann, San Mateo (1992)

Leighton, F.T., Maggs, B.M., Rao, S.B.: Packet routing and jobshop scheduling in O(congestion+dilation) steps. Combinatorica **14**(2), 167–180 (1994)

Leighton, F.T., Maggs, B.M., Richa, A.W.: Fast algorithms for finding O(congestion+dilation) packet routing schedules. Combinatorica **19**(3), 375–401 (1999)

Leighton, F.T., Makedon, F., Plotkin, S.A., Stein, C., Tardos, É., Tragoudas, S.: Fast approximation algorithms for multicommodity flow problems. J. Comp. Syst. Sci. **50**(2), 228–243 (1995)

Leighton, T., Rao, S.: An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms. In: Proceedings of the 29th Annual Symposium on Foundations of Computer Science, pp. 422–431, IEEE Computer Society (1988)

Leighton, T., Rao, S.: Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. J. ACM **46**(6), 787–832, 29th FOCS, 1988 (1999)

Leighton, T., Shor, P.: Tight bounds for minimax grid matching with applications to the average case analysis of algorithms. Combinatorica **9** 161–187 (1989)

Leiserson, C.E., Saxe, J.B.: Retiming synchronous circuitry. Algorithmica **6**, 5–35 (1991)

Lemke, C.E., Howson, J.T.: Equilibrium points of bimatrix games. J. Soc. Indust. Appl. Math. **12**, 413–423 (1964)

Lempel, A., Even, S., Cederbaum, I.: An algorithm for planarity testing of graphs. In: Rosentiehl, P. (ed.) Theory of Graphs: International Symposium. New York, Gordon and Breach, pp. 215–232 (1967)

Lenstra, A., Lenstra, H. (eds.): The Development of the Number Field Sieve. Lecture Notes in Mathematics, vol. 1544. Springer (1993)

Lenstra, A.K., Lenstra, H.W. Jr., Manasse, M.S., Pollard, J.M.: The number field sieve. In: Proceedings of the Twenty Second Annual ACM Symposium on Theory of Computing, Baltimore, Maryland, 14–16 May 1990, pp. 564–572

Lenstra, A.K., Lenstra, Jr., H.W., Lovász, L.: Factoring polynomials with rational coefficients. Math Ann. **261**, 513–534 (1982)

Lenstra, J.K., Shmoys, D., Tardos, E.: Approximation algorithms for scheduling unrelated parallel machines. Math. Program. **46**(3A), 259–271 (1990)

Lenstra Jr, H.W.: Solving the Pell equation. Not. Am. Math. Soc. **49**, 182–192 (2002)

Lenstra Jr., H.W.: Integer programming with a fixed number of variables. Math. Oper. Res. **8**(4), 538–548 (1983)

Leonardi, S.: A simpler proof of preemptive flow-time approximation. Approximation and On-line Algorithms. In: Bampis, E. (ed.) Lecture Notes in Computer Science. Springer, Berlin (2003)

Leonardi, S.: On-line network routing. In: Fiat, A., Woeginger, G. (eds.) Online Algorithms – The State of the Art. Chap. 11, pp. 242–267. Springer, Heidelberg (1998)

Leonardi, S., Raz, D.: Approximating total flow time on parallel machines. In: Proceedings of the Annual ACM Symposium on the Theory of Computing STOC, 1997, pp. 110–119

Leonardi, S., Vitaletti, A.: Randomized lower bounds for online path coloring. In: Proc. of the second International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM'98), pp. 232–247. (1998)

Leone, P., Rolim, J., Nikoletseas, S.: An Adaptive Blind Algorithm for Energy Balanced Data Propagation in Wireless Sensor Networks. In: Proc. of the IEEE International Conference on Distributed Computing in Sensor Networks (DCOSS). Lecture Notes in Computer Science (LNCS), vol. 3267, pp. 35–48. Springer (2005)

Leong, B., Liskov, B., Morris, R.: Geographic Routing without Planarization. In: 3rd Symposium on Networked Systems Design & Implementation (NSDI), San Jose, California, USA, May 2006

Leong, B., Mitra, S., Liskov, B.: Path Vector Face Routing: Geographic Routing with Local Face Information. In: 13th IEEE International Conference on Network Protocols (ICNP), Boston, Massachusetts, USA, November 2005

Leong, T., Shor, P., Stein, C.: Implementation of a combinatorial multicommodity flow algorithm. In: Johnson, D.S., McGeoch, C.C. (eds.) Network flows and matching. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 12, pp. 387–406. AMS, Providence (1991)

Leung, J., Whitehead, J.: On the complexity of fixed-priority scheduling of periodic, real-time tasks. Perform. Eval. **2**, 237–250 (1982)

Leutenegger, S.T., Lopez, M.A., Edington, J.: STR: A simple and efficient algorithm for R-tree packing. In: Proc. 13th IEEE International Conference on Data Engineering, 1997, pp. 497–506

Levcopoulos, C., Krznaric, D.: Quasi-Greedy Triangulations Approximating the Minimum Weight Triangulation. J. Algorithms **27**(2), 303–338 (1998)

Levcopoulos, C., Krznaric, D.: The Greedy Griangulation can be Computed from the Delaunay Triangulation in Linear Time. Comput. Geom. **14**(4), 197–220 (1999)

Levcopoulos, C., Lingas, A.: On Approximation Behavior of the Greedy Triangulation for Convex Polygons. Algorithmica **2**, 15–193 (1987)

Levcopoulos, C., Lingas, A.: There are planar graphs almost as good as the complete graphs and almost as cheap as minimum spanning trees. Algorithmica **8**(3), 251–256 (1992)

Levcopoulos, C., Narasimhan, G., Smid, M.: Improved algorithms for constructing fault-tolerant spanners. Algorithmica **32**, 144–156 (2002)

Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions, and reversals. Doklady Akademii Nauk SSSR **163**(4):845–848 (1965) (Russian). Soviet Physics Doklady **10**(8), 707–710 (1966) (English translation)

Levin, L.A.: Universal enumeration problems. Probl. Pereda. Inf. **9**(3), 115–116 (1973)

Levin, L.A.: Universal Search Problems. Проблемы передачи информации **9**(3), 265–266, (1973). In Russian. English translation in: Trakhtenbrot, B.A.: A Survey of Russian Approaches to Perebor (Brute-force Search) Algorithms. Annals of the History of Computing **6**(4), 384–400 (1984)

Li, B., Golin, M., Italiano, G., Deng, X., Sohraby, K.: On the optimal placement of web proxies in the internet. In: Proceedings of the 18th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), pp. 1282–1290. IEEE Computer Society, Los Alamitos (1999)

Li, C., Pion, S., Yap, C.K.: Recent progress in Exact Geometric Computation. J. Log. Algebr. Program. **64**(1), 85–111 (2004)

Li, J., Jannotti, J., De Couto, D.S.J., Karger, D.R., Morris, R.: A scalable location service for geographic ad hoc routing. In Proceedings of the Sixth International Conference on Mobile Computing and Networking, Boston, Massachusetts, Aug 2000, pp. 120–130

Li, L., Halpern, J.Y., Bahl, P., Wang, Y.-M., Wattenhofer, R.: Analysis of a cone-based distributed topology control algorithms for wireless multi-hop networks. In: PODC: ACM Symposium on Principle of Distributed Computing, Newport, 26–29 August 2001

Li, M., Chen, X., Li, X., Ma, B., Vitanyi, P.M.B.: The similarity metric. IEEE Trans. Inf. Theory **50**, 3250–3264 (2004)

Li, M., Lu, X.C., Peng, W.: Dynamic delaunay triangulation for wireless ad hoc network. In Proceedings of the Sixth International Workshop on Advanced Parallel Processing Technologies, Hong Kong, China, Oct 2005, pp. 382–389

Li, M., Ma, B., Wang, L.: Finding similar regions in many sequences. J. Comput. Syst. Sci. (1999)

Li, M., Ma, B., Wang, L.: Finding similar regions in many strings. In: Proceedings of the Thirty-first Annual ACM Symposium on Theory of Computing, pp. 473–482. Atlanta (1999)

Li, M., Ma, B., Wang, L.: On the Closest String and Substring Problems. J. ACM **49**(2), 157–171 (2002)

Li, M., Tromp, J., Vitányi, P.M.B.: How to share concurrent wait-free variables. J. ACM **43**(4), 723–746 (1996) (Preliminary version: Li, M., Vitányi, P.M.B. A very simple construction for atomic multiwriter register. Tech. Rept. TR-01–87, Computer Science Dept., Harvard University, Nov. 1987)

Li, M., Tromp, J., Zhang, L.: Some Notes on the Nearest Neighbour Interchange Distance. J. Theor. Biol. **26**(182), 463–467 (1996)

Li, M., Yao, A.C., Yao, F.F.: Discrete and Continuous Min-Energy Schedules for Variable Voltage Processors, Proceedings of the National Academy of Sciences USA, 103, pp. 3983–3987. National Academy of Science of the United States of America, Washington, DC, USA (2005)

Li, M., Yao, F.F.: An Efficient Algorithm for Computing Optimal Discrete Voltage Schedules. SIAM J. Comput. **35**(3), 658–671. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA (2005)

Li, N., Hou, J.C., Sha, L.: Design and analysis of a MST-based distributed topology control algorithm for wireless ad-hoc networks. In: 22nd Annual Joint Conference Of The IEEE Computer And Communications Societies (INFOCOM 2003), vol. 3, 1–3 April 2003, pp. 1702–1712

Li, Q., Rus, D.: Communication in disconnected ad hoc networks using message relay. Journal of Parallel and Distributed Computing (JPDC) **63**(1), 75–86 (2003). Special Issue on Wireless and Mobile Ad-hoc Networking and Computing, edited by A Boukerche

Li, W.-H., Gu, X.: The size distribution of insertions and deletions in human and rodent pseudogenes suggests the logarithmic gap penalty for sequence alignment. J. Mol. Evol. **40**, 464–473 (1995)

Li, X.-Y.: Approximate MST for UDG locally. In: COCOON, Big Sky, 25–28 July 2003

Li, X.-Y., Wan, P.-J.: Theoretically good distributed CDMA/OVSF code assignment for wireless ad hoc networks. In: Proceedings of 11th Internatioanl Computing and Combinatorics Conference (COCOON), Kunming, 16–19 August 2005

Li, X.-Y., Wan, P.-J., Wang, Y., Frieder, O.: Sparse power efficient topology for wireless networks. In: IEEE Hawaii Int. Conf. on System Sciences (HICSS), Big Island, 7–10 January 2002

Li, X.-Y., Wang, Y.: Efficient construction of low weighted bounded degree planar spanner. Int. J. Comput. Geom. Appl. **14**, 69–84 (2004)

Li, X.-Y., Wang, Y., Song, W.-Z., Wan, P.-J., Frieder, O.: Localized minimum spanning tree and its applications in wireless ad hoc networks. In: IEEE INFOCOM, Hong Kong, 7–11 March 2004

Li, X.Y.: Applications of computational geometry in wireless ad hoc networks. In: Cheng, X.Z., Huang, X., Du, D.Z. (eds.) Ad Hoc Wireless Networking, pp. 197–264. Kluwer, Dordrecht (2003)

Li, X.Y., Calinescu, G., Wan, P.J.: Distributed Construction of a Planar Spanner and Routing for Ad Hoc Wireless Networks. In: IEEE INFOCOM 2002, New York, NY, 23–27 June 2002

Li, X.Y., Teng, S.H., Üngör, A.: Biting: Advancing front meets sphere packing. Int. J. Num. Methods Eng. **49**(1–2), 61–81 (2000)

Li., W.-H.: Molecular Evolution. Sinauer, Sunderland (1997)

Libman, L., Orda, A.: Atomic Resource Sharing in Noncooperative Networks. Telecommun. Syst. **17**(4), 385-409 (2001)

Liefke, H., Suciu, D.: XMILL: An efficient compressor for XML data. In: Proceedings of the 2000 ACM SIGMOD Int. Conf. on Management of Data, pp. 153–164. ACM, New York, USA (2000)

Lifshits, Y., Mozes, S., Weimann, O., Ziv-Ukelson, M.: Speeding up HMM decoding and training by exploiting sequence repetitions. Algorithmica to appear doi:10.1007/s00453-007-9128-0

Lillis, J., Cheng, C.-K., Lin, T.-T.: Optimal and efficient buffer insertion and wire sizing. In: Proc. of Custom Integrated Circuits Conf., pp. 259–262. IEEE Press, Piscataway (1995)

Lin, G.H., Xue, G.: Signed genome rearrangements by reversals and transpositions: models and approximations. Theor. Comput. Sci. **259**, 513–531 (2001)

Lin, J.-H., Vitter, J.: ϵ-approximations with minimum packing constraint violation. In: 24th ACM STOC, pp. 771–782 (1992)

Lin, Y.-L., Huang, X., Jiang, T., Chao, K.-M.: MAVG: locating non-overlapping maximum average segments in a given sequence. Bioinformatics **19**, 151–152 (2003)

Lin, Y.-L., Jiang, T., Chao, K.-M.: Efficient algorithms for locating the length-constrained heaviest segments with applications to biomolecular sequence analysis. J. Comput. Syst. Sci. **65**, 570–586 (2002)

Lin, Y.L., Skiena, S.: Algorithms for Square Roots of Graphs. SIAM J. Discret. Math. **8**, 99–118 (1995)

Lingas, A.: Heuristics for minimum edge length rectangular partitions of rectilinear figures. In: Proc. 6th GI-Conference, Dortmund, January 1983. Springer

Lingas, A.: Subexponential-time algorithms for minimum weight triangulations and related problems. In: Proceedings 10th Canadian Conference on Computational Geometry (CCCG), McGill University, Montreal, Quebec, 10–12 August 1998

Lingas, A., Pinter, R.Y., Rivest, R.L., Shamir, A.: Minimum edge length partitioning of rectilinear polygons. In: Proc. 20th Allerton Conf. on Comm. Control and Compt., Illinos (1982)

Linial, N.: Finite metric-spaces—combinatorics, geometry and algorithms. In: Proceedings of the International Congress of Mathematicians, vol. III, Beijing, 2002, pp. 573–586. Higher Ed. Press, Beijing (2002)

Linial, N.: Locality in distributed graph algorithms. SIAM J. Comput. **21**(1), 193–201 (1992)

Linial, N., London, E., Rabinovich, Y.: The geometry of graphs and some of its algorithmic applications. Combinatorica **15**(2), 215–245 (1995). Also in Proc. 35th FOCS, pp. 577–591 (1994)

Linial, N., London, E., Rabinovich, Y.: The geometry of graphs and some of its algorithmic applications. In: IEEE Symposium on Foundations of Computer Science, pp. 577–591 (1994)

Linial, N., Mansour, Y., Nisan, N.: Constant depth circuits, Fourier transform and learnability. J. ACM **40**(3), 607–620 (1993)

Lipton, R., Rose, D., Tarjan, R.E.: Generalized nested dissection. SIAM. J. Numer. Anal. **16**, 346–358 (1979)

Lipton, R.J., Markakis, E., Mehta, A.: Playing large games using simple startegies. In: Proceedings of the 4th ACM Conference on Electronic Commerce (EC'03), pp. 36–41. San Diego, 9–13 June 2003

Lipton, R.J., Tarjan, R.E.: A separator theorem for planar graphs. SIAM J. Appl. Math. **36**(2), 177–189 (1979)

Lipton, R.J., Tarjan, R.E.: Applications of a planar separator theorem. SIAM J. Comput. **9**(3), 615–627 (1980)

Liskov, B.: Practical uses of synchronized clocks in distributed systems. Distrib. Comput. **6**, 211–219 (1993). Invited talk at the 9th Annual ACM Symposium on Principles of Distributed Computing, Quebec City 22–24 August 1990

Littlestone, N.: Learning quickly when irrelevant attributes abound: A new linear threshold algorithm. Mach. Learn. **2**(4), 285–318 (1988)

Littlestone, N., Warmuth, M.K.: The weighted majority algorithm. Inf. Comp. **108**(2), 212–261 (1994)

Liu, C., Layland, J.: Scheduling algorithms for multiprogramming in a hard real-time environment. J. ACM **20**, 46–61 (1973)

Liu, D., Prabhakaran, M.: On randomized broadcasting and gossiping in radio networks. In: Proc. 8th Annual International Computing Combinatorics Conference, pp. 340-349, Singapore (2002)

Liu, H.-F., Chao, K.-M.: Algorithms for Finding the Weight-Constrained *k* Longest Paths in a Tree and the Length-Constrained *k* Maximum-Sum Segments of a Sequence. Theoret. Comput. Sci. in revision (2008)

Liu, H.-F., Chao, K.-M.: On locating disjoint segments with maximum sum of densities. In: Proceedings of the 17th Annual International Symposium on Algorithms and Computation. LNCS, vol. 4288, pp. 300–307 (2006)

Liu, H., Wong, D.F.: Network-Flow-based Multiway Partitioning with Area and Pin Constraints. IEEE Trans. CAD Integr. Circuits Syst. **17**(1), 50–59 (1998)

Lo, H.-K.: Classical communication cost in distributed quantum information processing – a generalization of quantum communication complexity. Phys. Rev. A **62**, 012313 (2000)

Lo, H.-K., Chau, H.F.: Unconditional security of quantum key distribution over arbitrarily long distances. Science **283**, 2050–2056 (1999)

Lo, W.-K., Hadzilacos, V.: Using failure detectors to solve consensus in asynchronous shared memory systems. In: Proceedings of the 8th International Workshop on Distributed Algorithms, LNCS 857, pp. 280–295, September 1994

Lockhart, P.J., Steel, M.A., Hendy, M.D., Penny, D.: Recovering evolutionary trees under a more realistic model of sequence evolution. Mol. Biol. Evol. **11**, 605–612 (1994)

Lomonosov, M.: Bernoulli Scheme with Closure. Probl. Inf. Transm. **10**, 73–81 (1974)

Lopez-Ortiz, A.: On-Line Target Searching in Bounded and Unbounded Domains: Ph. D. Thesis, Technical Report CS-96-25, Dept. of Computer Sci., Univ. of Waterloo (1996)

Lopez-Ortiz, A., Schuierer, S.: The Ultimate Strategy to Search on m Rays? Theor. Comput. Sci. **261**(2), 267–295 (2001)

Lopresti, D.P., Tomkins, A.: Block Edit Models for Approximate String Matching. Theoretical. Comput. Sci. **181**(1), 159–179 (1997)

Lothaire, M. (ed.): Algebraic Combinatorics on Words. Cambridge University Press, Cambridge (2002)

Lothaire, M. (ed.): Applied Combinatorics on Words. Cambridge University Press, Cambridge (2005)

Lotker, Z., Patt-Shamir, B.: Nearly optimal FIFO buffer management for DiffServ. In: Proceedings of the 21st ACM Symposium on Principles of Distributed Computing (PODC 2002), pp. 134–142. ACM, New York (2002)

Lotker, Z., Patt-Shamir, B.: Nearly optimal FIFO buffer management for two packet classes. Comput. Netw. **42**(4), 481–492 (2003)

Lotker, Z., Patt-Shamir, B., Pavlov, E., Peleg, D.: Minimum-weight spanning tree construction in $O(\log\log n)$ communication rounds. SIAM J. Comput. **35**(1), 120–131 (2005)

Lotker, Z., Patt-Shamir, B., Peleg, D.: Distributed MST for constant diameter graphs. Distrib. Comput. **18**(6), 453–460 (2006)

Loubal, P.: A network evaluation procedure. Highway Res. Rec. **205**, 96–109 (1967)

Lovász, L.: On Determinants, Matchings and Random Algorithms. In: Budach, L. (ed.) Fundamentals of Computation Theory, FCT'79, pp. 565–574. Akademie-Verlag, Berlin (1979)

Lovász, L.: On the ratio of optimal integral and fractional covers. Discret. Math. **13**, 383–390 (1975)

Lovász, L.: On the Shannon capacity of a graph. IEEE Trans. Inf. Theor. **25**, 2–13 (1979)

Lovász, L., Plummer, M.D.: Matching Theory. Akadémiai Kiadó – North Holland, Budapest (1986)

Lu, C., Alvarez, G.A., Wilkes, J.: Aqueduct:online data migration with performance guarantees. In: Proceedings of the Conference on File and Storage Technologies (2002)

Lu, H.-I., Ravi, R.: Approximating maximum leaf spanning trees in almost linear time. J. Algorithm **29**, 132–141 (1998)

Lu, H.-I., Yeh, C.-C.: Balanced parentheses strike back. Accepted to ACM Trans. Algorithms (2007)

Luby, M.: A Simple Parallel Algorithm for the Maximal Independent Set Problem. SIAM J. Comput. **15**, 1036–1053 (1986)

Luby, M.: Removing randomness in parallel without processor penalty. J. Comput. Syst. Sci. **47**(2), 250–286 (1993)

Luccio, F., Pagli, L.: On the upper bound on the rotation distance of binary trees. Inf. Process. Lett. **31**(2), 57–60 (1989)

Luchangco, V., Moir, M., Shavit, N.: On the uncontended complexity of consensus. In: Proc. 17th Annual International Symposium on Distributed Computing, 2005

Lueker, G.S.: An average-case analysis of bin packing with uniformly distributed item sizes. Tech. Rep. Report No 181, Dept. of Information and Computer Science, University of California, Irvine, CA (1982)

Lund, C., Yannakakis, M.: On the hardness of approximating minimization problems. J. ACM **41**(5), 960–981 (1994)

Lund, C., Yannakakis, M.: The approximation of maximum subgraph problems. In: Proc. 20th ICALP. LNCS, vol. 700, pp. 40–51. Springer, Berlin (1993)

Lundelius, J., Lynch, N.: A new fault-tolerant algorithm for clock synchronization. Inf. Comput. **77**, 1–36 (1988)

Lunter, G., Miklós, I., Drummond, A., Jensen, J., Hein, J.: Bayesian coestimation of phylogeny and sequence alignment. BMC Bioinformatics (2005)

Lunter, G., Miklós, I., Drummond, A., Jensen, J., Hein, J.: Bayesian phylogenetic inference under a statistical indel model. In: Lecture Notes in Bioinformatics, Proceedings of WABI2003, vol. 2812, pp. 228–244 (2003)

Lunter, G.A., Miklós, I., Song, Y.S., Hein, J.: An efficient algorithm for statistical multiple alignment on arbitrary phylogenetic trees. J. Comp. Biol. **10**(6), 869–889 (2003)

Luo, J., Hubaux, J.-P.: Joint Mobility and Routing for Lifetime Elongation in Wireless Networks. In: Proc. 24th INFOCOM (2005)

Luo, J., Panchard, J., Piórkowski, M., Grossglauser, M., Hubaux, J.P.: Mobiroute: Routing towards a mobile sink for improving lifetime in sensor networks. In: Gibbons, P.B., Abdelzaher, T., Aspnes, J., Rao, R. (eds.) 2nd IEEE/ACM International Conference on Distributed Computing in Sensor Systems (DCOSS 2005). Lecture Notes in Computer Science (LNCS), vol. 4026, pp 480–497. Springer, Berlin (2006)

Lusena, C., Goldsmith, J., Mundhenk, M.: Nonapproximability results for partially observable markov decision processes. J. Artif. Intell. Res. **14**, 83–103 (2001)

Lynch, N.A.: Distributed Algorithms. Morgan Kaufmann, CA (1996)

Lyngsø, R.B.: Complexity of pseudoknot prediction in simple models. In: Proceedings of the 31th International Colloquium on Automata, Languages and Programming (ICALP), 2004, pp. 919–931

Lyngsø, R.B., Pedersen, C.N.S.: RNA pseudoknot prediction in energy based models. J. Comput. Biol. **7**, 409–428 (2000)

Lyngsø, R.B., Zuker, M., Pedersen, C.N.S.: Fast evaluation of internal loops in RNA secondary structure prediction. Bioinformatics **15**, 440–445 (1999)

Ma, B.: A polynomial time approximation scheme for the closest substring problem. In: Proc. 11th Annual Symposium on Combinatorial Pattern Matching, Montreal, pp. 99–107. (2000)

Ma, B., Tromp, J., Li, M.: PatternHunter: Faster and More Sensitive Homology Search. Bioinformatics **18**, 440–445 (2002)

Ma, B., Zhang, K., Lajoie, G., Doherty-Kirby, A., Hendrie, C., Liang, C., Li, M.: PEAKS: Powerful software for peptide de novo sequencing by tandem mass spectrometry. Rapid Commun. Mass Spectrom. **17**(20), 2337–2342 (2003)

Ma, B., Zhang, K., Liang, C.: An effective algorithm for the peptide de novo sequencing from MS/MS spectrum. J. Comput. Syst. Sci. **70**, 418–430 (2005)

Ma, Y., Plotkin, S.: Improved lower bounds for load balancing of tasks with unknown duration. Inf. Process. Lett. **62**, 31–34 (1997)

Maaß, M.G., Nowak, J.: Text indexing with errors. In: Proceedings of Symposium on Combinatorial Pattern Matching, 2005, pp. 21–32

MacWilliams, F.J., Sloane, N.J.A.: The Theory of Error–Correcting Codes. North–Holland, Amsterdam (1977)

MacWilliams, F.J., Sloane, N.J.A.: The Theory of Error Correcting Codes. North-Holland, Amsterdam (1981)

Madduri, K., Bader, D., Berry, J., Crobak, J.: Parallel shortest path algorithms for solving large-scale instances. In: 9th DIMACS Implementation Challenge Workshop: Shortest Paths, DIMACS Center, Piscataway, NJ, 13–14 Nov 2006

Maekawa, M.: A $\sqrt{n}$ algorithm for mutual exclusion in decentralized systems. ACM Trans. Comput. Syst. **3**(2), 145–159 (1985)

Magazine, M.J., Oguz, O.: A fully polynomial approximation algorithm for the 0–1 knapsack problem. Eur. J. Oper. Res. **8**, 270–273 (1981)

Maggs, B.M., Miller, G.L., Parekh, O., Ravi, R., Woo, S.L.M.: Finding effective support-tree preconditioners. In: Symposium on Parallel Algorithms and Architectures, pp. 176–185 (2005)

Magniez, F.: Multi-linearity self-testing with relative error. Theory Comput. Syst. **38**(5), 573–591 (2005)

Magniez, F., Nayak, A.: Quantum complexity of testing group commutativity. Algorithmica **48**(3), 221–232 (2007) Preliminary version in Proc. ICALP (2005) quant-ph/0506265

Magniez, F., Nayak, A.: Quantum complexity of testing group commutativity. In: Proceedings of the International Colloquium Automata, Languages and Programming (ICALP'05), 2005, pp. 1312–1324

Magniez, F., Nayak, A., Roland, J., Santha, M.: Search via quantum walk. quant-ph/0608026. In: Proc. of 39th ACM Symp. on Theory of Computing (STOC), San Diego, 11–13 June, pp. 575–584 (2007)

Magniez, F., Santha, M., Szegedy, M.: Quantum algorithms for the triangle problem. SIAM J. Comput. **37**(2), 413–424 (2007) Preliminary version in Proc. SODA 2005

Mahajan, M., Raman, V.: Parameterizing above Guaranteed Values: MAXSAT and MAXCUT. J. Algorithms **31**(2), 335–354 (1999)

Mahajan, R., Hariharan, R.: Derandomizing semidefinite programming based approximation algorithms. In: Proceedings of the 36th Annual IEEE Symposium on Foundations of Computer Science (FOCS), Milwaukee 1995, pp. 162–169

Mahdian, M.: Facility Location and the Analysis of Algorithms through Factor-Revealing Programs. Ph. D. thesis, MIT, Cambridge (2004)

Mahdian, M.: Random popular matchings. In: Proceedings of the 7th ACM Conference on Electronic Commerce (EC), pp. 238–242 Venice, July 10–14 2006

Mahdian, M., Pál, M.: Universal facility location. In: European Symposium on Algorithms, pp. 409–421. Budapest, Hungary, September 16–19 2003

Mahdian, M., Pál, M.: Universal facility location. In: Proceedings of the 11th Annual European Symposium on Algorithms (ESA). Lecture Notes in Computer Science, vol. 2832, pp. 409–421. Springer, Berlin (2003)

Mahdian, M., Ye, Y., Zhang, J.: Approximation algorithms for metric facility location problems. SIAM J. Comput. **36**(2), 411–432 (2006)

Maheshwari, N., Sapatnekar, S.S.: Efficient retiming of large circuits, IEEE Transactions on Very Large-Scale Integrated Systems. **6**, 74–83 (1998)

Main, M.G.: Detecting leftmost maximal periodicities. Discret. Appl. Math. **25**, 145–153 (1989)

Main, M.G., Lorentz, R.J.: An $O(n \log n)$ algorithm for finding all repetitions in a string. J. Algorithms **5**(3), 422–432 (1984)

Mäkinen, E.: On the rotation distance of binary trees. Inf. Process. Lett. **26**(5), 271–272 (1988)

Mäkinen, V., Navarro, G.: Dynamic Entropy-Compressed Sequences and Full-Text Indexes. In: Proc. 17th Symposium on Combinatorial Pattern Matching (CPM). LNCS, vol. 4009, pp. 307–318. Springer, Berlin (2006)

Mäkinen, V., Navarro, G.: Succinct suffix arrays based on run-length encoding. Nord. J. Comput. **12**(1), 40–66 (2005)

Makinen, V., Navarro, G., Ukkonen, E.: Approximate matching of run-length compressed strings. Algorithmica **35**(4), 347–369 (2003)

Mäkinen, V., Navarro, G., Ukkonen, E.: Approximate Matching of Run-Length Compressed Strings. In: Proc. 12th Symposium on Combinatorial Pattern Matching (CPM'01). LNCS, vol. 2089, pp. 31–49 (2001)

Mäkinen, V., Ukkonen, E.: Local Similarity Based Point-Pattern Matching. In: Proc. 13th Annual Symposium on Combinatorial Pattern Matching (CPM 2002). LNCS, vol. 2373, pp. 115–132. Springer, Berlin (2002)

Malhotra, V.S.: On the stability of multiple partner stable marriages with ties. In: Proceedings of ESA '04: the 12th Annual European Symposium on Algorithms. Lecture Notes in Computer Science, vol. 3221, pp. 508–519. Springer, Berlin (2004)

Malik, S., Wang, A.R., Brayton, R.K., Sangiovanni-Vincentelli, A.: Logic Verification using Binary Decision Diagrams in a Logic Synthesis Environment. In: IEEE International Conference on Computer-Aided Design, pp. 6–9. (1988)

Malkhi, D., Naor, M., Ratajczak, D.: Viceroy: A scalable and dynamic emulation of the butterfly. In: Proceedings of the 21st ACM Symposium on Principles of Distributed Computing (PODC '02), 2002, pp. 183–192

Malkhi, D., Reiter, M.: Byzantine quorum systems. Distrib. Comput. **11**(4), 203–213 (1998)

Malkhi, D., Reiter, M., Wool, A., Wright, R.: Probabilistic quorum systems. Inf. Comput. J. **170**, 184–206 (2001)

Malkhi, D., Reiter, M.K.: An architecture for survivable coordination in large-scale systems. IEEE Trans. Knowl. Data Engineer. **12**, 187–202 (2000)

Malony, A., Reed, D.: Visualizing Parallel Computer System Performance. In: Simmons, M., Koskela, R., Bucher, I. (eds.) Instrumentation for Future Parallel Computing Systems. ACM Press, New York (1989) pp. 59–90

Manasse, M., McGeoch, L.A., Sleator, D.: Competitive algorithms for online problems. In: Proc. 20th Symp. Theory of Computing (STOC), pp. 322–333. ACM (1988)

Manasse, M.S., McGeoch, L.A., Sleator, D.D.: Competitive algorithms for on-line problems. In: Proceeding 20th Annual ACM Symposium on the Theory of Computing, pp. 322–333, Chicago, IL (1988)

Manasse, M.S., McGeoch, L.A., Sleator, D.D.: Competitive algorithms for server problems. J. Algorithms **11**, 208–230 (1990)

Manber, U.: A text compression scheme that allows fast searching directly in the compressed file. ACM Trans. Inf. Syst. **15**(2), 124–136 (1997)

Manber, U., Myers, G.: Suffix arrays: a new method for on-line string searches. SIAM J. Comput. **22**(5), 935–948 (1993)

Mandoiu, I.I., Vazirani, V.V., Ganley, J.L.: A new heuristic for rectilinear Steiner trees. In: Proc. Intl. Conf. on Computer-Aided Design, San Jose, (1999)

Maniscalco, M.A., Puglisi, S.J.: Faster lightweight suffix array construction. In: Proc. 17th Australasian Workshop on Combinatorial Algorithms, pp. 16–29. Univ. Ballavat, Ballavat (2006)

Manku, G.S., Bawa, M., Raghavan, P.: Symphony: Distributed hashing in a small world. In: Proc. 4th USENIX Symposium on Internet Technologies and Systems (USITS 2003) 2003, pp. 127–140

Manlove, D., Sng, C.: Popular matchings in the capacitated house allocation problem. In: Proceedings of the 14th Annual European Symposium on Algorithms (ESA), pp. 492–503 (2006)

Manlove, D.F.: private communication (2006)

Manlove, D.F.: Stable marriage with ties and unacceptable partners. Technical Report TR-1999-29, University of Glasgow, Department of Computing Science, January (1999)

Manlove, D.F., Irving, R.W., Iwama, K., Miyazaki, S., Morita, Y.: Hard variants of stable marriage. Theor. Comput. Sci. **276**(1–2), 261–279 (2002)

Manne, A.S.: Plant location under economies-of-scale – decentralization and computation. Manag. Sci. **11**, 213–235 (1964)

Manolopoulos, Y., Nanopoulos, A., Papadopoulos, A.N., Theodoridis, Y.: R-trees: Theory and Applications. Springer, London (2005)

Mansour, Y.: Randomized interpolation and approximation of sparse polynomials. SIAM J. Comput. **24**, 357–368 (1995)

Mansour, Y., Patt-Shamir, B., Lapid, O.: Optimal smoothing schedules for real-time streams. In: Proc. 19th Symp. on Principles of Distributed Computing (PODC), pp. 21–29. ACM, New York (2000)

Mansour, Y., Sahar, S.: Implementation Issues in the Fourier Transform Algorithm. Mach. Learn. **40**(1), 5–33 (2000)

Manzini, G.: An analysis of the Burrows–Wheeler transform. J. ACM **48**, 407–430 (2001)

Manzini, G.: Two space saving tricks for linear time LCP array computation. In: Proc. 9th Scandinavian Workshop on Algorithm Theory. LNCS, vol. 3111, pp. 372–383. Springer, Berlin/Heidelberg (2004)

Manzini, G., Ferragina, P.: Engineering a lightweight suffix array construction algorithm. Algorithmica **40**, 33–50 (2004)

Marathe, M.V., Breu, H., Hunt III, H.B., Ravi, S.S., Rosenkrantz, D.J.: Simple Heuristics for Unit Disk Graphs. Networks **25**, 59–68 (1995)

Marathe, V., Scherer, W., Scott, M.: Adaptive software transactional memory. In: Proc. 19th Annual International Symposium on Distributed Computing, 2005

Marathe, V.J., Moir, M.: Toward high performance nonblocking software transactional memory. In: Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming. pp. 227–236, ACM, New York, USA (2008)

Marczewski, E.: Sur deux propriétés des classes d' ensembles. Fund. Math. **33**, 303–307 (1945)

Marple, D.P.: Performance Optimization of Digital VLSI Circuits. Technical Report CSL-TR-86-308, Stanford University, October 1986

Marple, D.P.: Transistor Size Optimization in the Tailor Layout System. In: Proceedings of the 26th ACM/IEEE Design Automation Conference, pp. 43–48. June 1989

Martello, S., Toth, P. Knapsack Problems: Algorithms and Computer Implementations. Wiley, Chichester (1990)

Martin, R.K., Sethares, W.A., Williamson, R.C., Johnson, Jr., C.R.: Exploiting sparsity in adaptive filters. IEEE Trans. Signal Process. **50**(8), 1883–1894 (2002)

Marx, D.: The Closest Substring problem with small distances. In: Proceedings of the 46th FOCS, pp 63–72. IEEE Press, (2005)

Mas-Colell, A., Whinston, M.D., Green, J.R.: Microeconomic Theory. Oxford University Press, Oxford (1995)

Masek, W., Paterson, M.: A faster algorithm for computing string edit distances. J. Comput. Syst. Sci. **20**, 18–31 (1980)

Maskin, E.S.: Auctions, development, and privatization: Efficient auctions with liquidity-constrained buyers. Eur. Econ. Rev. **44**(4–6), 667–681 (2000)

Massey, J.L., Mathys, P.: The collision channel without feedback. IEEE Trans. Inf. Theor. **31**, 192–204 (1985)

Mathews, D.H., Sabina, J., Zuker, M., Turner, D.H.: Expanded sequence dependence of thermodynamic parameters improves prediction of RNA secondary structure. J. Mol. Biol. **288**, 911–940 (1999)

Matias, Y., Şahinalp, C.: On the optimality of parsing in dynamic dictionary based data compression. In: Proceedings 10th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '99), pp. 943–944 (1999)

Matias, Y., Segal, E., Vitter, J.S.: Efficient bundle sorting. SIAM J. Comput. **36**(2), 394–410 (2006)

Matousek, J.: Lectures on Discrete Geometry. Springer, New York (2002)

Matsumoto: Competitive Analysis of the Round Robin Algorithm. in: 3rd International Symposium on Algorithms and Computation, 1992, pp. 71–77

Mattern, F.: Virtual time and global states of distributed systems. In: Cosnard, M., Quinton, P. (eds.) Parallel and Distributed Algorithms, pp.215–226. North-Holland, Amsterdam (1989)

Mattle, K., Weinfurter, H., Kwiat, P.G., Zeilinger, A.: Dense coding in experimental quantum communication. Phys. Rev. Lett. **76**, 4656–4659 (1996)

Mavrides, M.: Triangular arbitrage in the foreign exchange market – inefficiencies, technology and investment opportunities. Quorum Books, London (1992)

Mavronicolas, M., Spirakis, P.G.: The price of selfish routing. In: Proc. on 33rd Annual ACM Symposium on Theory of Computing (STOC), pp. 510–519. ACM, Heraklion (2001)

Mayers, D.: Quantum key distribution and string oblivious transfer in noisy channels. In: Advances in Cryptology –CRYPTO '96. Lecture Notes in Computer Science, vol. 1109, pp. 343–357. Springer (1996)

Maymounkov, P., Mazières, D.: Kademlia: A peer-to-peer information system based on the XOR metric. In: Proc. 1st Intl. Workshop on Peer-to-Peer Systems (IPTPS 2002), 2002, pp. 53–65

Mazoit, F.: The branch-width of circular-arc graphs. In: 7th Latin American Symposium on Theoretical Informatics (LATIN 2006), 2006, pp. 727–736

McCaskill, J.S.: The equilibrium partition function and base pair binding probabilities for RNA secondary structure. Biopolymers **29**, 1105–1119 (1990)

McCluskey, E.J.: Minimization of Boolean functions. Bell Syst. Tech. J. **35**(6), 1417–1444 (1956)

McCreight, E.M.: A space-economical suffix tree construction algorithm. J. ACM **23**, 262–272 (1976)

McCreight, E.M.: Priority search trees. SIAM J. Comput. **14**, 257–276 (1985)

McGeoch, C.: A bibliography of algorithm experimentation. In: Data Struktures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 59, 251–254. American Mathematical Society, Providence, RI (2002)

McGeoch, C.C.: Toward an experimental method for algorithm simulation. INFORMS J. Comp. **1**(1), 1–15 (1996)

McGeoch, L., Sleator, D.: A strongly competitive randomized paging algorithm. Algorithmica **6**(6), 816–825 (1991)

McGlynn, M.J., Borbash, S.A.: Birthday Protocols for Low Energy Deployment and Flexible Neighborhood Discovery in Ad Hoc Wireless Networks. In: Proc. of the 2nd ACM Int. Symposium on Mobile Ad Hoc Networking & Computing (MOBIHOC), (2001)

McIlroy, P.M., Bostic, K., McIlroy, M.D.: Engineering radix sort. Comput. Syst. **6**, 5–27 (1993)

McKay, B.D.: Hadamard equivalence via graph isomorphism. Discret. Math. **27**, 213–214 (1979)

McKay, B.D.: Practical graph isomorphism. Congr. Numer. **30**, 45–87 (1981)

McKay, B.D., Meynert, A., Myrvold, W.: Small Latin squares, quasigroups and loops. J. Comb. Des. **15**, 98–119 (2007)

McMillan, K.L.: Symbolic Model Checking. Kluwer Academic Publishers (1993)

McMorris, F.R.: On the compatibility of binary qualitative taxonomic characters. Bull. Math. Biol. **39**, 133–138 (1977)

McNaughton, R., Yamada, H.: Regular expressions and state graphs for automata. IRE Trans. Elect. Comput. **9**(1), 39–47 (1960)

McVitie, D., Wilson, L.B.: The stable marriage problem. Commun. ACM **14**, 486–490 (1971)

Mead, C.A., Conway, L.: Introduction to VLSI Systems. Addison-Wesley, (1980)

Mecke, S., Wagner, D.: Solving geometric covering problems by data reduction. In: Proceedings of the 12th Annual European Symposium on Algorithms (ESA '04). LNCS, vol. 3221, pp. 760–771. Springer, Berlin (2004)

Megiddo, N.: Computational Complexity and the Game Theory Approach to Cost Allocation for a Tree. Math. Oper. Res. **3**, 189–196 (1978)

Megiddo, N.: Cost allocation for Steiner trees. Networks **8**(1), 1–6 (1978)

Megiddo, N., Papadimitriou, C.H.: On total functions, existence theorems and computational complexity. Theor. Comp. Sci. **81**, 317–324 (1991)

Mehlhorn, K.: A best possible bound for the weighted path length of binary search trees. SIAM J. Comput. **6**(2), 235–239 (1977)

Mehlhorn, K.: Data Structures and Algorithms 1: Sorting and Searching. EATCS Monographs on Theoretical Computer Science, vol. 1. Springer, Berlin (1984)

Mehlhorn, K.: Dynamic binary search. SIAM J. Comput. **8**(2), 175–198 (1979)

Mehlhorn, K., Mutzel, P., Näher, S.: An implementation of the hopcroft and tarjan planarity test. Tech. Rep. MPI-I-93-151, Saarbrücken (1993)

Mehlhorn, K., Näher, S.: LEDA: A Platform for Combinatorial and Geometric Computing. Cambridge University Press, Cambridge (1999)

Mehlhorn, K., Näher, S.: LEDA: A Platform for Combinatorial and Geometric Computing. Commun. ACM. **38**(1), 96–102 (1995)

Mehlhorn, K., Sanders, P.: Scanning multiple sequences via cache memory. Algorithmica **35**, 75–93 (2003)

Melideo, G., Varricchio, S.: Learning unary output two-tape automata from multiplicity and equivalence queries. In: ALT '98. Lecture Notes in Computer Science, vol. 1501, pp. 87–102. Springer, Berlin (1998)

Mellor-Crummey, J.M., Scott, M.L.: Algorithms for scalable synchronization on shared-memory multiprocessors. ACM Trans. Comput. Syst. **9**(1), 21–65 (1991)

Mendel, M., Naor, A.: Ramsey partitions and proximity data structures. J. Eur. Math. Soc. **9**(2), 253–275 (2007)

Mendelson, R., Tarjan, R., Thorup, M., Zwick, U.: Melding priority queues. ACM TALG **2**(4), 535–556 (2006). Announced at SODA'04

Mendonca, D., Raghavachari, M.: Comparing the efficacy of ranking methods for multiple round-robin tournaments. Eur. J. Oper. Res. **123**, 593–605 (1999)

Menezes, A., van Oorschot, P., Vanstone, S.: Handbook of Applied Cryptography. CRC Press, Boca Raton (1997)

Message Passing Interface Forum. MPI: A message-passing interface standard. Technical report, University of Tennessee, Knoxville, TN, June 1995. Version 1.1

Messer, P.W., Arndt, P.F.: The majority of recent short DNA insertions in the human genome are tandem duplications. Mol. Biol. Evol. **24**(5), 1190–7 (2007)

Mestre, J.: Weighted popular matchings. In: Proceedings of the 16th International Colloquium on Automata, Languages, and Programming (ICALP), pp. 715–726 (2006)

Metzler, D., Fleißner, R., Wakolbringer, A., von Haeseler, A.: Assessing variability by joint sampling of alignments and mutation rates. J. Mol. Evol. **53**, 660–669 (2001)

Meyer auf der Heide, F., Vöcking, B.: Shortest-Path Routing in Arbitrary Networks. J. Algorithms **31**(1), 105–131 (1999)

Meyer, U., Sanders, P., Sibeyn, J.F. (eds.): Algorithms for Memory Hierarchies. LNCS, vol. 2625. Springer, Berlin (2003)

Meyerson, A.: Online facility location. In: Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp. 426–431. IEEE Computer Society, Los Alamitos (2001)

Micali, S., Vazirani, V.V.: An $O(\sqrt{V}E)$ Algorithm for Finding Maximum Matching in General Graphs. In: Proceedings of the 21st Annual IEEE Symposium on Foundations of Computer Science (FOCS), 1980, pp. 17–27

Micciancio, D., Goldwasser, S.: Complexity of Lattice Problems: A Cryptographic Perspective. The Kluwer International Series in Engineering and Computer Science, vol. 671. Kluwer Academic Publishers, Boston, Massachusetts (2002)

Michael, M., Scott, M.: Nonblocking algorithms and preemption-safe locking on multiprogrammed shared memory multiprocessors. J. Parall. Distrib. Comput. **51**(1), 1–26 (1998)

Michel, R., Corentin, T.: In search of the holy grail: Looking for the weakest failure detector for wait-free set agreement. Technical Report TR 06-1811, INRIA, August 2006

Miklós, I., Meyer, I.M., Nagy, B.: Moments of the Boltzmann distribution for RNA secondary structures. Bull. Math. Biol. **67**, 1031–1047 (2005)

Milchtaich, I.: Congestion games with player-specific payoff functions. Games Econ. Behav. **13**, 111–124 (1996)

Milenkovic, V.J.: Densest translational lattice packing of non-convex polygons. Proc. 16th ACM Annual Symp. on Computational Geometry, 280–289 (2000)

Miller, G., Naor, J.: Flow in planar graphs with multiple sources and sinks. SIAM J. Comput. **24**, 1002–1017 (1995)

Miller, G.L.: Finding small simple cycle separators for 2-connected planar graphs. J. Comput. Syst. Sci. **32**, 265–279 (1986)

Miller, G.L., Reif, J.H.: Parallel tree contraction and its applications. In: Proceedings of the 26th Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp. 478–489 (1985)

Miller, G.L., Teng, S.H., Vavasis, S.A.: An unified geometric approach to graph separators. In: Proc. 32nd Annu. IEEE Sympos. Found. Comput. Sci. 1991, pp. 538–547

Miller, W., Myers, E.W.: Sequence comparison with concave weighting functions. Bull. Math. Biol. **50**(2), 97–120 (1988)

Mills, D.L.: Computer Network Time Synchronization: The Network Time Protocol. CRC Press, Boca Raton (2006)

Miltersen, P.B.: Cell probe complexity – a survey. In: 19th Conference on the Foundations of Software Technology and Theoretical Computer Science (FSTTCS), Advances in Data Structures Workshop, 1999

Miltersen, P.B.: Lower bounds for Union-Split-Find related problems on random access machines. In: 26th ACM Symposium on Theory of Computing (STOC), 1994, pp. 625–634

Miltersen, P.B., Nisan, N., Safra, S., Wigderson, A.: On data structures and asymmetric communication complexity. J. Comput. Syst. Sci. **57**(1), 37–49 (1998). See also STOC'95

Miltersen, P.B., Subramanian, S., Vitter, J.S., Tamassia, R.: Complexity models for incremental computation. In: Ausiello, G., Italiano, G.F. (eds.) Special Issue on Dynamic and On-line Algorithms. Theor. Comp. Sci. **130**(1), 203–236 (1994)

Min, M., Du, H., Jia, X., Huang, X., Huang, C.-H., Wu, W.: Improving construction for connected dominating set with Steiner tree in wireless sensor networks. J. Glob. Optim. **35**, 111–119 (2006)

Min, M., Huang, S.C.-H., Liu, J., Shragowitz, E., Wu, W., Zhao, Y., Zhao, Y.: An Approximation Scheme for the Rectilinear Steiner Minimum Tree in Presence of Obstructions. Fields Inst. Commun. **37**, 155–164 (2003)

Minsky, M., Papert, S.: Perceptrons. MIT Press, Cambridge (1969)

Minsky, M., Papert, S.: Perceptrons: An Introduction to Computational Geometry. The MIT Press, (1969)

Mirchandani, P.B., Francis, R.L.: Discrete Location Theory. Wiley, New York (1990)

Mirrokni, V.S.: Approximation Algorithms for Distributed and Selfish Agents. Ph.D. thesis, Massachusetts Institute of Technology (2005)

Mishchenko, A., Chatterjee, S., Brayton, R., Ciesielski, M.: An integrated technology mapping environment. International Workshop on Logic Synthesis (2005)

Mitchell, D., Selman, B., Levesque, H.: Hard and easy distribution of sat problems. In: 10th National Conference on Artificial Intelligence, pp. 459–465. AAAI Press, Menlo Park (1992)

Mitchell, J.: A Geometric Shortest Path Problem, with Application to Computing a Longest Common Subsequence in Run-Length Encoded Strings. Technical Report, Dept. of Applied Mathematics, SUNY Stony Brook (1997)

Mitchell, J.S.B.: Guillotine subdivisions approximate polygonal subdivisions: A simple new method for the geometric *k*-MST problem. In: Proc. 7th ACM-SIAM Symposium on Discrete Algorithms, 1996, pp. 402–408.

Mitchell, J.S.B.: Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric TSP, *k*-MST, and related problems. SIAM J. Comput. **28**(4), 1298–1309 (1999)

Mitchell, J.S.B.: Guillotine subdivisions approximate polygonal subdivisions: Part II – A simple polynomial-time approximation scheme for geometric *k*-MST, TSP, and related problem. SIAM J. Comput. **29**(2), 515–544 (1999)

Mitchell, J.S.B.: Guillotine subdivisions approximate polygonal subdivisions: Part III – Faster polynomial-time approximation scheme for geometric network optimization, manuscript, State University of New York, Stony Brook (1997)

Mitchell, J.S.B., Blum, A., Chalasani, P., Vempala, S.: A constant-factor approximation algorithm for the geometric *k*-MST problem in the plane. SIAM J. Comput. **28**(3), 771–781 (1999)

Mitchell, T.: Machine Learning. McGraw Hill (1997)

Miyazaki, M., Shinohara, A., Takeda, M.: An improved pattern matching algorithm for strings in terms of straight-line programs. J. Discret. Algorithms **1**(1), 187–204 (2000)

Miyazaki, T.: The complexity of McKay's canonical labelling algorithm. In: Groups and Computation, II. DIMACS Ser. Discret. Math. Theor. Comput. Sci., vol. 28, pp. 239–256. American Mathematical Society, Providence, RI (1997)

Mo, Y.-Y., Chu, C.: A hybrid dynamic/quadratic programming algorithm for interconnect tree optimization. IEEE Trans. Comput. Des. **20**(5), 680–686 (2001)

Moffat, A.: An improved data structure for cumulative probability tables. Softw. Prac. Exp. **29**, 647–659 (1999)

Moffat, A., Anh, V.N.: Binary codes for locally homogeneous sequences. Inf. Process. Lett. **99**(5), 75–80 (2006) Source code available from www.cs.mu.oz.au/~alistair/rbuc/

Moffat, A., Stuiver, L.: Binary interpolative coding for effective index compression. Inf. Retr. **3**(1), 25–47 (2000)

Moffat, A., Turpin, A.: Compression and Coding Algorithms. Kluwer Academic Publishers, Boston (2002)

Mohar, B., Poljak, S.: Eigenvalues and the max-cut problem. Czechoslov Math. J. **40**(115), 343–352 (1990)

Molly, M., Reed, B.: Graph Coloring and the Probabilistic method. Springer (2002)

Monasson, R., Zecchina, R.: Statistical mechanics of the random *k*-sat problem. Phys. Rev. E **56**, 1357–1361 (1997)

Monderer, D., Shapley, L.: Potential games. Games Econ. Behav. **14**, 124–143 (1996)

Monien, B.: How to find long paths efficiently. Ann. Discret. Math. **25**, 239–254 (1985)

Monien, B., Speckenmeyer, E.: Solving satisfiability in less than 2^n steps. Discret. Appl. Math. **10**, 287–295 (1985)

Monma, C.L., Shallcross, D.F.: Methods for designing communications networks with certain two-connected survivability constraints. Operat. Res. **37**(4), 531–541 (1989)

Mony, H., Baumgartner, J., Paruthi, V., Kanzelman, R., Kuehlmann, A.: Scalable Automated Verification via Expert-System Guided Transformations. In: Formal Methods in CAD. (2004)

Moore, C., Russell, A.: Quantum walks on the hypercube. In: Proc. RANDOM (2002)

Moore, G.W., Goodman, M., Barnabas, J.: An iterative approach from the standpoint of the additive hypothesis to the dendrogram problem posed by molecular data sets. J. Theor. Biol. **38**, 423–457 (1973)

Mor, T., Roychowdhury, V., Lloyd, S., Fernandez, J.M., Weinstein, Y.: Algorithmic cooling. US Patent 6,873,154 (2005)

Moret, B.: Towards a discipline of experimental algorithmics. In: Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 59, 197–214. American Mathematical Society, Providence, RI (2002)

Moret, B.M.E., Bader, D.A., Warnow, T.: High-performance algorithm engineering for computational phylogenetics. J. Supercomput. **22**, 99–111 (2002) Special issue on the best papers from ICCS'01

Moret, B.M.E., Bader, D.A., Warnow, T., Wyman, S.K., Yan, M.: GRAPPA: a highperformance computational tool for phylogeny reconstruction from gene-order data. In: Proc. Botany, Albuquerque, August 2001

Moret, B.M.E., Shapiro, H.D.: Algorithms and experiments: The new (and old) methodology. J. Univers. Comput. Sci. **7**(5), 434–446 (2001)

Moret, B.M.E., Shapiro, H.D.: An empirical assessment of algorithms for constructing a minimum spanning tree. In: DIMACS Ser. Discrete Math. Theoret. Comput. Sci., vol. 15, Am. Math. Soc., Providence, RI (1994)

Moret, B.M.E., Tang, J., Warnow, T.: Reconstructing phylogenies from gene-content and gene-order data. In: Gascuel, O. (ed.) Mathematics of Evolution and Phylogeny. pp. 321–352, Oxford Univ. Press, USA (2005)

Moret, B.M.E., Wyman, S., Bader, D.A., Warnow, T., Yan, M.: A new implementation and detailed study of breakpoint analysis. In: Proc. 6th Pacific Symp. Biocomputing (PSB 2001), pp. 583–594, Hawaii, January 2001

Morrison, M., Brillhart, J.: A method of factoring and the factorization of F_7

Mosca, M., Ekert, A.: The Hidden Subgroup Problem and Eigenvalue Estimation on a Quantum Computer. In: Proceedings 1st NASA International Conference on Quantum Computing & Quantum Communications. Lecture Notes in Computer Science, vol. 1509, pp. 174–188. Springer, London (1998)

Moscibroda, T.: Locality, Scheduling, and Selfishness: Algorithmic Foundations of Highly Decentralized Networks. Doctoral Thesis Nr. 16740, ETH Zurich (2006)

Moscibroda, T., von Rickenbach, P., Wattenhofer, R.: Analyzing the Energy-Latency Trade-off during the Deployment of Sensor Networks. In: Proc. of the 25th Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), (2006)

Moscibroda, T., Wattenhofer, R.: Coloring Unstructured Radio Networks. In: Proc. of the 17th ACM Symposium on Parallel Algorithms and Architectures (SPAA), pp. 39–48 (2005)

Moscibroda, T., Wattenhofer, R.: Maximal Independent Sets in Radio Networks. In: Proc. of the 23rd ACM Symposium on Principles of Distributed Computing (PODC), pp. 148–157 (2005)

Moses, Y., Shimony, B.: A new proof of the GHS minimum spanning tree algorithm. In: Distributed Computing, 20th Int. Symp. (DISC), Stockholm, Sweden, September 18–20, 2006. Lecture Notes in Computer Science, vol. 4167, pp. 120–135. Springer, Berlin Heidelberg (2006)

Moshkov, M.Y.: Conditional tests. Probl. Kibern. (in Russian) **40**, 131–170 (1983)

Mossel, E., O'Donnell, R., Servedio, R.A.: Learning functions of *k* relevant variables. J. Comp. Syst. Sci. **69**(3), 421–434 (2004)

Mostefaoui, A., Rajsbaum, S., Raynal, M.: The Combined Power of Conditions and Failure Detectors to Solve Asynchronous Set Agreement. In: Proc. 24th ACM Symposium on Principles of Distributed Computing (PODC'05), pp. 179–188. ACM Press, New York (2005)

Mostéfaoui, A., Rajsbaum, S., Raynal, M., Roy, M.: Condition-based consensus solvability: a hierarchy of conditions and efficient protocols. Distrib. Comput. **17**, 1–20 (2004)

Mostefaoui, A., Raynal, M.: *k*-Set Agreement with Limited Accuracy Failure Detectors. In: Proc. 19th ACM Symposium on Principles of Distributed Computing, pp. 143–152. ACM Press, New York (2000)

Mostefaoui, A., Raynal, M.: Randomized Set Agreement. In: Proc. 13th ACM Symposium on Parallel Algorithms and Architectures (SPAA'01), Hersonissos (Crete) pp. 291–297. ACM Press, New York (2001)

Motwani, R., Phillips, S., Torng, E.: Non-Clairvoyant Scheduling. Theor. Comput. Sci. **130**(1), 17–47 (1994)

Motwani, R., Raghavan, P.: Randomized Algorithms. Cambridge University Press, New York (1995)

Moulin, H.: Incremental cost sharing: Characterization by coalition strategy-proofness. Social Choice and Welfare, **16**, 279–320 (1999)

Moulin, H., Shenker, S.: Strategyproof sharing of submodular costs: budget balance versus efficiency. Econ. Theor. **18**(3), 511–533 (2001)

Mu'alem, A., Nisan, N.: Truthful approximation mechanisms for restricted combinatorial auctions. In: Proc. 18th Nat. Conf. Artificial Intelligence, pp. 379–384. AAAI (2002)

Mu'alem, A., Schapira, M.: Setting lower bounds on truthfulness. In: Proc. 18th Symposium on Discrete Algorithms (SODA), 2007

Mucha, M., Sankowski, P.: Maximum Matchings in Planar Graphs via Gaussian Elimination. Algorithmica **45**, 3–20 (2006)

Mucha, M., Sankowski, P.: Maximum Matchings via Gaussian Elimination. In: Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS), 2004 pp. 248–255

Müller-Hannemann, M., Schnee, M.: Finding all attractive train connections by multi-criteria pareto search. In: Geraets, F., Kroon, L.G., Schöbel, A., Wagner, D., Zaroliagis, C.D. (eds.) Algorithmic Methods for Railway Optimization, International Dagstuhl Workshop, Dagstuhl Castle, Germany, June 20–25, 2004, 4th International Workshop, ATMOS 2004, Bergen, September 16–17, 2004, Revised Selected Papers, Lecture Notes in Computer Science, vol. 4359, pp. 246–263. Springer, Berlin (2007)

Müller-Hannemann, M., Schnee, M.: Paying less for train connections with MOTIS. In: Kroon, L.G., Möhring, R.H. (eds.) Proceedings of the 5th Workshop on Algorithmic Methods and Models for Optimization of Railways (ATMOS'05), Dagstuhl, Germany, Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany 2006. Dagstuhl Seminar Proceedings, no. 06901

Müller-Hannemann, M., Schulz, F., Wagner, D., Zaroliagis, C.D.: Timetable information: Models and algorithms. In: Geraets, F., Kroon, L.G., Schöbel, A., Wagner, D., Zaroliagis, C.D. (eds.) Algorithmic Methods for Railway Optimization, International Dagstuhl Workshop, Dagstuhl Castle, Germany, June 20–25, 2004, 4th International Workshop, ATMOS 2004, Bergen, September 16–17, 2004, Revised Selected Papers, Lecture Notes in Computer Science, vol. 4359, pp. 67–90. Springer (2007)

Mulmuley, K., Vazirani, U.V., Vazirani, V.V.: Matching is as easy as matrix inversion. In: Proceedings of the 19th Annual ACM Conference on Theory of Computing, pp. 345–354. ACM Press, New York (1987)

Mulzer, W., Rote, G.: Minimum-weight triangulation is NP-hard. In: Proceedings 22nd Annual ACM Symposium on Computational Geometry, SoCG'06, Sedona, AZ, USA. ACM Press, New York, NY, USA (2006)

Mundell, R.A.: Currency areas, exchange rate systems, and international monetary reform, paper delivered at Universidad del CEMA, Buenos Aires, Argentina. http://www.robertmundell.net/pdf/Currency (2000). Accessed 17 Apr 2000

Mundell, R.A.: Gold Would Serve into the 21st Century. Wall Street Journal, 30 September 1981, pp. 33

Munkres, J.: Algorithms for the assignment and transportation problems. J. Soc. Ind. Appl. Math. **5**, 32–38 (1957)

Munro, I.: Tables. In: Proc. 16th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS). LNCS, vol. 1180, Hyderabad, 18–20 December, pp. 37–42 (1996)

Munro, J.I., Raman, R., Raman, V., Rao, S.S.: Succinct representations of permutations. In: Proceedings of the 30th International Colloquium on Automata, Languages and Programming (ICALP). Lecture Notes in Computer Science (LNCS), vol. 2719, pp. 345–356. Springer, Berlin (2003)

Munro, J.I., Raman, V.: Succinct representation of balanced parentheses and static trees. SIAM J. Comput. **31**(3), 762–776 (2001)

Munro, J.I., Raman, V., Rao, S.S.: Space efficient suffix trees. J. Algorithms **39**(2), 205–222 (2001)

Munro, J.I., Raman, V., Storm, A.J.: Representing dynamic binary trees succinctly. In: Rao Kosaraju, S. (ed.) Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, pp. 529–536, Philadelphia (2001)

Munro, J.I., Rao, S.S.: Succinct representations of functions. In: Díaz, J., Karhumäki, J., Lepistö, A., Sannella, D. (eds.): Proceedings of the 31st International Colloquium on Automata, Languages and Programming, pp. 1006–1015. Springer, Heidelberg (2004)

Munro, J.I., Rao, S.S.: Succinct representations of functions. In: Proceedings of the International Colloquium on Automata, Languages and Programming (ICALP). Lecture Notes in Computer Science (LNCS), vol. 3142, pp. 1006–1015. Springer, Berlin (2004)

Munro, J.I., Srinivasa Rao, S.: Succinct representation of data structures. In: Mehta, D., Sahni, S. (eds.) Handbook of Data Structures with Applications, Chap 37. Chapman and Hall/CRC Press (2005)

Murata, H., Fujiyoshi, K., Nakatake, S., Kajitani, Y.: A solution space of size $(n!)^2$ for optimal rectangle packing. In: 8th Karuizawa Workshop on Circuits and Systems, April 1995, pp. 109–114

Murata, H., Nakatake, S., Fujiyoshi, K., Kajitani, Y.: VLSI Module placement based on rectangle-packing by Sequence-Pair. IEEE Trans. Comput. Aided Design (TCAD) **15**(12), 1518–1524 (1996)

Murchland, J.: The effect of increasing or decreasing the length of a single arc on all shortest distances in a graph. Technical report, LBS-TNT-26, London Business School, Transport Network Theory Unit, London (1967)

Murgai, R., Brayton, R.K., Sangiovanni-Vincentelli, A.: On clustering for minimum delay/area. In: Proceedings of IEEE International Conference on Computer-Aided Design, 1991, pp. 6–9

Murota, K.: Discrete Convex Analysis. Math. Program. **83**, 313–371 (1998)

Murota, K.: Discrete Convex Analysis. Soc. Ind. Appl. Math. Philadelphia (2003)

Murphy, W., et al.: Dynamics of Mammalian Chromosome Evolution Inferred from Multispecies Comparative Maps. Science **309**, 613–617 (2005)

Muthukrishnan, S.: Data streams: Algorithms and applications. Found. Trends Theor. Comput. Sci. **1**, pp.1–126 (2005)

Muthukrishnan, S., Rajaraman, R., Shaheen, A., Gehrke, J.: Online Scheduling to Minimize Average Stretch. SIAM J. Comput. **34**(2), 433–452 (2004)

Muthukrishnan, S., Sahinalp, S.C.: Approximate nearest neighbors and sequence comparison with block operations. Proc. ACM STOC 416–424 (2000)

Myers, B.: Taxonomies of Visual Programming and Program Visualization. J. Visual Lang. Comp. **1**, 97–123 (1990)

Myers, E.: A four Russians algorithm for regular expression pattern matching. J. ACM **39**(2), 430–448 (1992)

Myers, E.G.: A sublinear algorithm for approximate keyword searching. Algorithmica **12**, 345–374 (1994)

Myers, E.W.: Approximate matching of network expressions with spacers. J. Comput. Biol. **3**(1), 33–51 (1996)

Myers, E.W., Miller, W.: Approximate matching of regular expressions. Bullet. Math. Biol. **51**, 7–37 (1989)

Myers, E.W., Miller, W.: Optimal Alignments in Linear Space. Bioinformatics **4**, 11–17 (1988)

Myers, G.: A fast bit-vector algorithm for approximate string matching based on dynamic progamming. J. ACM **46**(3), 395–415 (1999)

Myerson, R.B.: Optimal auction design. Math. Oper. Res. **6**, 58–73 (1981)

Na, J.: Linear-time construction of compressed suffix arrays using $o(n \log n)$-bit working space for large alphabets. In: Proc. 16th Symposium on Combinatorial Pattern Matching (CPM). LNCS, vol. 3537, pp. 57–67. Springer, Berlin (2005)

Na, J.C., Giancarlo, R., Park, K.: $O(n^2 \log n)$ time on-line construction of two-dimensional suffix trees. In: Proceedings of the 11th International Computing and Combinatorics Conference, 2005, pp. 273–282

Nachtigall, K.: Time depending shortest-path problems with applications to railway networks. Eur. J. Oper. Res. **83**, 154–166 (1995)

Nagamochi, H.: An improved bound on the one-sided minimum crossing number in two-layered drawings. Discret. Comput. Geom. **33**, 569–591 (2005)

Nagamochi, H., Ibaraki, T.: A linear-time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph. Algorithmica **7**(5–6), 583–596 (1992)

Nagel, W.E., Arnold, A., Weber, M., Hoppe, H.C., Solchenbach, K.: VAMPIR: visualization and analysis of MPI resources. Supercomputer 63. **12**(1), 69–80 (1996)

Nakatake, S., Fujiyoshi, K., Murata, H., Kajitani, Y.: Module packing based on the BSG-structure and IC layout applications. IEEE TCAD **17**(6), 519–530 (1998)

Nakatake, S., Murata, H., Fujiyoshi, K., Kajitani, Y.: Bounded Sliceline Grid (BSG) for module packing. IEICE Technical Report, October 1994, VLD94-66, vol. 94, no. 313, pp. 19–24 (in Japanese)

Nakhleh, L., Warnow, T., Linder, C.R.: Reconstructing reticulate evolution in species – theory and practice. In: Proc. 8th Annual International Conference on Research in Computational Molecular Biology (RECOMB 2004), 2004, pp. 337–346

Nakhleh, L., Warnow, T., Ringe, D., Evans, S.N.: A comparison of phylogenetic reconstruction methods on an Indo-European dataset. Trans. Philol. Soc. **103**, 171–192 (2005)

Naor, J., Naor, M.: Small-bias probability spaces: efficient constructions and applications. SIAM J. Comput. Comput. **22**(4), 838–856 (1993)

Naor, J.S., Zosin, L.: A 2-Approximation Algorithm for the Directed Multiway Cut Problem. SIAM J. Comput. **31**(2), 477–492 (2001). Preliminary version in FOCS 1997

Naor, M., Reingold, O.: Number-theoretic constructions of efficient pseudo-random functions. J. ACM **51**(2), 231–262 (2004)

Naor, M., Stockmeyer, L.: What can be computed locally? In: Proc. of the 25th Annual ACM Symposium on Theory of Computing (STOC), pp. 184–193 (1993)

Naor, M., Wieder, U.: Novel architectures for p2p applications: the continuous-discrete approach. In: The Fifteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '03), 2003

Naor, M., Wool, A.: The load, capacity and availability of quorum systems. SIAM J. Comput. **27**, 423–447 (1998)

Narasimhan, G., Smid, M.: Geometric spanner networks. Cambridge University Press, New York (2006)

Narasimhan, G., Smid, M.: Geometric Spanner Networks. Cambridge University Press, New York (2007)

Narasimhan, G., Zachariasen, M.: Geometric Minimum Spanning Trees via Well-Separated Pair Decompositions. ACM J. Exp. Algorithms **6**, 6 (2001)

Nash, J.F.: Equilibrium point in n-person games. In: Proceedings of the National Academy of the USA, vol. 36, issue 1, pp. 48–49 (1950)

Nash, J.F.: Equilibrium point in n-person games. Proc. Natl. Acad. Sci. USA **36**(1), 48–49 (1950)

Nash, J.F.: Non-cooperative games. Ann. Math. **54**, 268–295 (1951)

Navarra, A.: 3-d minimum energy broadcasting. In: Proceedings of the 13th Colloquium on Structural Information and Communication Complexity (SIROCCO), pp. 240–252 (2006)

Navarra, A.: Tighter bounds for the minimum energy broadcasting problem. In: Proceedings of the 3rd International Symposium on Modeling and Optimization in Mobile, Ad-hoc and Wireless Networks (WiOpt), pp. 313–322 (2005)

Navarro, G.: A guided tour to approximate string matching. ACM Comput. Surv. **33**(1), 31–88 (2001)

Navarro, G.: Approximate regular expression searching with arbitrary integer weights. Nord. J. Comput. **11**(4), 356–373 (2004)

Navarro, G.: Indexing text using the Ziv–Lempel trie. J. Discret. Algorithms **2**, 87–114 (2004)

Navarro, G.: Nr-grep: a fast and flexible pattern matching tool. Softw. Pr. Exp. **31**, 1265–1312 (2001)

Navarro, G.: Regular expression searching on compressed text. J. Discret. Algorithms **1**(5–6), 423–443 (2003)

Navarro, G., Baeza-Yates, R.: A hybrid indexing method for approximate string matching. J. Discret. Algorithms **1**, 21–49 (2000)

Navarro, G., Baeza-Yates, R.: Very fast and simple approximate string matching. Inf. Proc. Lett. **72**, 65–70 (1999)

Navarro, G., Baeza-Yates, R.A., Sutinen, E., Tarhio, J.: Indexing methods for approximate string matching. IEEE Data Eng. Bull. **24**(4), 19–27 (2001)

Navarro, G., Chávez, E.: A metric index for approximate string matching. Theor. Comput. Sci. **352**(1–3), 266–279 (2006)

Navarro, G., Mäkinen, V.: Compressed full text indexes. ACM Comput. Surv. **39**(1) (2007)

Navarro, G., Paredes, R.: Practical construction of metric *t*-spanners. In: Proceedings of the 5th Workshop on Algorithm Engineering and Experiments, pp. 69–81, 11 January 2003. SIAM Press, Baltimore

Navarro, G., Raffinot, M.: Fast and flexible string matching by combining bit-parallelism and suffix automata. ACM J. Exp. Algorithm **5**, 4 (2000)

Navarro, G., Raffinot, M.: Fast and simple character classes and bounded gaps pattern matching, with applications to protein searching. J. Comput. Biol. **10**(6), 903–923 (2003)

Navarro, G., Raffinot, M.: Flexible Pattern Matching in Strings – Practical on-line search algorithms for texts and biological sequences. Cambridge University Press, Cambridge (2002)

Navarro, G., Raffinot, M.: New techniques for regular expression searching. Algorithmica **41**(2), 89–116 (2004)

Navarro, G., Tarhio, J.: LZgrep: A Boyer–Moore string matching tool for Ziv–Lempel compressed text. Softw. Pract. Exp. **35**(12), 1107–1130 (2005)

Nayak, A., Vishwanath, A.: Quantum walk on the line. quant-ph/0010117

Needleman, S.B., Wunsch, C.D.: A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins. J. Mol. Biol. **48**, 443–453 (1970)

Nekrutenko, A., Li, W.H.: Assessment of compositional heterogeneity within and between eukaryotic genomes. Genome Res. **10**, 1986–1995 (2000)

Nemhauser, G.L., Trotter, L.E.: Vertex packing: structural properties and algorithms. Math. Program. **8**, 232–248 (1975)

Nemhauser, G.L., Wolsey, L.A.: Integer and Combinatorial Optimization. Wiley, Hoboken (1999)

Nemhauser, G.L., Wolsey, L.A.: Integer and Combinatorial Optimization. Wiley, New York (1988)

Nemhauser, G.L., Wolsey, L.A.: Integer and Combinatorial Optimization. Wiley, New York (1990)

Nenakhov, E., Primak, M.: About one algorithm for finding the solution of the Arrow-Debreu Model. Kibernetica **3**, 127–128 (1983)

Newman, A.: A note on polyhedral relaxations for the maximum cut problem (2004). Unpublished manuscript

Neyman, J.: Molecular studies of evolution: a source of novel statistical problems. In: Gupta, S.S., Yackel, J. (eds) Statistical Decision Theory and Related Topics, pp. 1–27. Academic Press, New York (1971)

Ng, A.Y.: Feature selection, L_1 vs. L_2 regularization, and rotational invariance. In: Greiner, R., Schuurmans, D. (eds.) Proceedings of the 21st International Conference on Machine Learning, pp 615–622. The International Machine Learning Society, Princeton (2004)

Ng, A.Y., Coates, A., Diel, M., Ganapathi, V., Schulte, J., Tse, B., Berger, E., Liang, E.:Inverted autonomous helicopter flight via reinforcement learning. In: International Symposium on Experimental Robotics, 2004

Ng, C., Hirschberg, D.S.: Lower bounds for the stable marriage problem and its variants. SIAM J. Comput. **19**, 71–77 (1990)

Ng, M.P., Wormald, N.C.: Reconstruction of rooted trees from subtrees. Discrete Appl. Math. **69**(1–2), 19–31 (1996)

Ng, R.T., Han, J.: Efficient and effective clustering methods for spatial data mining. In: Proc. Symp. on Very Large Data Bases (VLDB), pp. 144–155. Santiago de Chile, 12–15 September 1994

Nguyen, P., Stern, J.: The two faces of lattices in cryptology. In: J. Silverman (ed.) Cryptography and lattices conference – CaLC 2001, Providence, RI, USA, March 2001. Lecture Notes in Computer Science, vol. 2146, pp. 146–180. Springer, Berlin (2001)

Nieberg, T., Hurink, J.L.: A PTAS for the Minimum Dominating Set Problem in Unit Disk Graphs. LNCS, vol. 3879, pp. 296–306. Springer, Berlin (2006)

Niedermeier, R.: Invitation to Fixed-Parameter Algorithms. Oxford Lecture Series in Mathematics and Its Applications, vol. 31. Oxford University Press, Oxford (2006)

Niedermeier, R., Rossmanith, P.: New upper bounds for maximum satisfiability. J. Algorithms **26**, 63–88 (2000)

Nielsen, M.A., Chuang, I.L.: Quantum Computation and Quantum Information. Cambridge University Press, Cambridge (2000)

Nielsen, M.A., Knill, E., Laflamme, R.: Complete quantum teleportation using nuclear magnetic resonance. Nature **396**(6706), 52–55 (1998)

Nikoletseas, S., Chatzigiannakis, I., Antoniou, A., Efthymiou, C., Kinalis, A., Mylonas, G.: Energy Efficient Protocols for Sensing Multiple Events in Smart Dust Networks. In: Proc. 37th Annual ACM/IEEE Simulation Symposium (ANSS'04), pp. 15–24, IEEE Computer Society Press (2004)

Nikoletseas, S., Palem, K., Spirakis, P., Yung, M.: Connectivity Properties in Random Regular Graphs with Edge Faults. In: Special Issue on Randomized Computing of the International Journal of Foundations of Computer Science (IJFCS), vol. 11 no. 2, pp. 247–262, World Scientific Publishing Company (2000)

Nikoletseas, S., Palem, K., Spirakis, P., Yung, M.: Short Vertex Disjoint Paths and Multiconnectivity in Random Graphs: Reliable Network Computing. In: Proc. 21st International Colloquium on Automata, Languages and Programming (ICALP), pp. 508–515. Jerusalem (1994)

Nikoletseas, S., Pantziou, G., Psycharis, P., Spirakis, P.: On the reliability of fat-trees. In: Proc. 3rd International European Conference on Parallel Processing (Euro-Par), pp. 208–217, Passau, Germany (1997)

Nikoletseas, S., Raptopoulos, C., Spirakis, P.: Expander Properties and the Cover Time of Random Intersection Graphs. In: Proc of the 32nd MFCS, pp. 44–55. Springer, Berlin/Heidelberg (2007)

Nikoletseas, S., Raptopoulos, C., Spirakis, P.: The existence and Efficient construction of Large Independent Sets in General Random Intersection Graphs. In: Proc. of the 31st ICALP. LNCS, vol. 3142, pp. 1029–1040. Springer, Berlin/Heidelberg (2004)

Nikoletseas, S., Spirakis, P.: Expander Properties in Random Regular Graphs with Edge Faults. In: Proc. 12th Annual Symposium on

Theoretical Aspects of Computer Science (STACS), pp.421–432, München (1995)

Nisan, N. Ronen, A.: Algorithmic mechanism design. In: Proceedings of the 31st Annual ACM Symposium on Theory of Computing (STOC-99), pp. 129–140. Association for Computing Machinery, New York (1999)

Nisan, N.: Bidding and Allocation in Combinatorial Auctions. In: Proceedings of EC'00, pp. 1–12. Minneapolis, 17–20 October 2000

Nisan, N., Ronen, A.: Algorithmic mechanism design. Game. Econ. Behav. **35**, 166–196 (2001)

Nisan, N., Ronen, A.: Algorithmic mechanism design. In: Proc. 31st Annual Symposium on Theory of Computing (STOC99), Atlanta, 1–4 May 1999, pp. 129–140 (1999)

Nisan, N., Ronen, A.: Computationally feasible vcg mechanisms. In: Proc. of the 2nd ACM Conference on Electronic Commerce (EC'00), 2000

Nishimura, N., Ragde, P., Szeider, S.: Detecting backdoor sets with respect to Horn and binary clauses. In: Informal proceedings of SAT 2004, 7th International Conference on Theory and Applications of Satisfiability Testing, Vancouver, BC, Canada, 10–13 May 2004, pp. 96–103

Nishimura, N., Ragde, P., Szeider, S.: Solving SAT using vertex covers. Acta Inf. **44**(7–8), 509–523 (2007)

Nodine, M.H., Vitter, J.S.: Deterministic distribution sort in shared and distributed memory multiprocessors. In: Proceedings of the ACM Symposium on Parallel Algorithms and Architectures, June–July 1993, vol. 5, pp. 120–129, ACM Press, New York (1993)

Nodine, M.H., Vitter, J.S.: Greed Sort: An optimal sorting algorithm for multiple disks. J. ACM **42**(4), 919–933 (1995)

Noga, J., Seiden, S.S.: An optimal online algorithm for scheduling two machines with release times. Theor. Comput. Sci. **268**(1), 133–143 (2001)

Novikoff, A. B. J.: On convergence proofs on perceptrons. In: Proceedings of the Symposium on the Mathematical Theory of Automata, volume XII, pp. 615–622, (1962)

Nussinov, R., Jacobson, A.B.: Fast algorithm for predicting the secondary structure of single-stranded RNA. Proc. Natl. Acad. Sci. USA **77**, 6309–6313 (1980)

Nussinov, R., Pieczenik, G., Griggs, J., Kleitman, D.: Algorithms for loop matchings. SIAM J. Appl. Math. **35**, 68–82 (1978)

Nutt, G.: Operating System Projects Using Windows NT. Addison-Wesley, Reading (1999)

O'Donnell, R., Servedio, R.: Learning monotone decision trees in polynomial time. In: Proceedings of the 21st Conference on Computational Complexity (CCC), pp. 213–225, Prague, 16–20 July 2006

Ogurtsov, A.Y., Shabalina, S.A., Kondrashov, A.S., Roytberg, M.A.: Analysis of internal loops within the RNA secondary structure in almost quadratic time. Bioinformatics **22**, 1317–1324 (2006)

Ohnishi, H., Seki, H., Kasami, T.: A polynomial time learning algorithm for recognizable series. IEICE Transactions on Information and Systems, **E77-D(10)**(5), 1077–1085 (1994)

Ohta, T.: Near-neutrality in evolution of genes and gene regulation. Proc. Natl. Acad. Sci. USA **99**, 16134–16137 (2002)

Okanohara, D., Sadakane, K.: Practical entropy-compressed rank/select dictionary. In: Proc. 9th ACM-SIAM Workshop on Algorithm Engineering and Experiments (ALENEX '07), SIAM, to appear (2007)

Olariu, S., Stojmenovic, I.: Design guidelines for maximizing lifetime and avoiding energy holes in sensor networks with uniform distribution and uniform reporting. In: IEEE INFOCOM, Barcelona, Spain, April 24–25 2006

Olken, F.: Random Sampling from Databases. Ph. D. thesis, Department of Computer Science, University of California, Berkeley (1993)

Olmstead, R.G., Palmer, J.D.: Chloroplast DNA systematics: a review of methods and data analysis. Am. J. Bot. **81**, 1205–1224 (1994)

OpenMP Architecture Review Board. OpenMP: A proposed industry standard API for shared memory programming. www.openmp.org, October 1997

Ostrovsky, R., Patt-Shamir, B.: Optimal and efficient clock synchronization under drifting clocks. In: Proceedings of the 18th Annual Symposium on Principles of Distributed Computing, pp. 3–12, Atlanta, May (1999)

Ostrovsky, R., Rabani, Y.: Universal $O(\text{congestion} + \text{dilation} + \log^{1+\varepsilon} N)$ Local Control Packet Switching Algorithm. In: Proceedings of The Twenty-Ninth ACM Symposium on Theory of Computing, pp. 644–653 (1997)

Otten, R.H.J.M.: Automatic Floorplan Design. In: Proceedings of the 19th Design Automation Conference, pp. 261–267 (1982)

Ouchi, K.: Real/Expr: Implementation of an exact computation package. Master's thesis, New York University, Department of Computer Science, Courant Institute, January (1997). URL http://cs.nyu.edu/exact/doc/

Oum, S.I., Seymour, P.: Approximating clique-width and branch-width. J. Combin. Theor. Ser. B **96**, 514–528 (2006)

Owen, G.: On the Core of Linear Production Games. Math. Program. **9**, 358–370 (1975)

Ozery-Flato, M., Shamir, R.: Two notes on genome rearrangement. J. Bioinf. Comput. Biol. **1**, 71–94 (2003)

Page, L., Brin, S., Motwani, R., Winograd, T.: The PageRank Citation Ranking: Bringing Order to the Web. In: Technical Report. Stanford University, Stanford (1998)

Pagh, A., Pagh, R., Thorup, M.: On adaptive integer sorting. In: Proc. 12th ESA, 2004, pp. 556–579

Pagh, R.: A trade-off for worst-case efficient dictionaries. Nord. J. Comput. **7**, 151–163 (2000). See also SWAT'00

Pagh, R.: Low redundancy in static dictionaries with constant query time. SIAM J. Comput. **31**, 353–363 (2001)

Pagh, R.: Low redundancy in static dictionaries with $O(1)$ lookup time. In: Proceedings of ICALP '99. LNCS, vol. 1644, pp. 595–604. Springer, Berlin (1999)

Pagh, R.: On the cell probe complexity of membership and perfect hashing. In: Proceedings of the 33rd Annual ACM Symposium on Theory of Computing (STOC '01), pp. 425–432. ACM Press, New York (2001)

Pagh, R., Rodler, F.F.: Cuckoo hashing. J. Algorithms **51**, 122–144 (2004)

Pál, M., Tardos, É., Wexler, T.: Facility location with nonuniform hard capacities. In: Proceedings of the 42nd Annual Symposium on Foundations of Computer Science, pp. 329–338. Las Vegas, 14–17 October 2001

Palmer, J.D.: Chloroplast and mitochondrial genome evolution in land plants. In: Herrmann, R. (ed.) Cell Organelles, pp. 99–133. Springer, Vienna (1992)

Palmer, J.D., Herbon, L.A.: Tricircular mitochondrial genomes of Brassica and Raphanus: reversal of repeat configurations by inversion. Nucleic Acids Res. **14**, 9755–9764 (1986)

Pan, P.: Continuous retiming: Algorithms and applications. In: Proc. Intl. Conf. Comput. Design, pp. 116–121. IEEE Press, Los Almitos (1997)

Pan, P., Karandikar, A.K., Liu, C.L.: Optimal clock period clustering for sequential circuits with retiming. IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. **17**, 489–498 (1998)

Pan, P., Lin, C.C.: A New Retiming-based Technology Mapping Algorithm for LUT-based FPGAs. ACM International Symposium on Field-Programmable Gate Arrays (1998)

Pan, P., Liu, C.L.: Area Minimization for Floorplans. In: IEEE Trans. Comput. Aided Des. **14**(1), 123–132 (1995)

Pan, P., Liu, C.L.: Optimal Clock Period FPGA Technology Mapping for Sequential Circuits. ACM Trans. on Des. Autom. of Electron. Syst., **3**(3), 437–462 (1998)

Pan, P., Liu, C.L.: Optimal Clock Period FPGA Technology Mapping for Sequential Circuits. ACM/IEEE Design Automation Conference, June (1996)

Pan, P., Shi, W., Liu, C.L.: Area Minimization for Hierarchical Floorplans. In: Algorithmica **15**(6), 550–571 (1996)

Panagiotou, K., Souza, A.: On adequate performance measures for paging. In: STOC '06: Proceedings of the thirty-eighth annual ACM symposium on Theory of computing, pp. 487–496. ACM Press, New York, NY, USA (2006)

Panagopoulou P., Spirakis P.: Algorithms for pure Nash Equilibrium in weighted congestion games. ACM J. Exp. Algorithms **11**, 2.7 (2006)

Panaite, P., Pelc, A.: Exploring unknown undirected graphs. J. Algorithms **33**, 281–295 (1999)

Panigrahy, R.: Efficient hashing with lookups in two memory accesses. In: Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '05), pp. 830–839. SIAM, Vancouver, 23–25 January 2005

Papa, D.A., Markov, I.L.: Hypergraph partitioning and clustering. In: Gonzalez, T. (ed.) Handbook of algorithms. Taylor & Francis Group, Boca Raton, CRC Press, pp. 61–1 (2007)

Papadimitriou, C.: Computational Complexity. Addison-Wesley, Reading (1994)

Papadimitriou, C., Steiglitz, K.: Combinatorial Optimization: Algorithms and Complexity. Prentice-Hall, Englewood Cliffs (1982)

Papadimitriou, C., Yannakakis, M.: Linear programming without the matrix. In: Proc. of the 25th ACM Symposium on Theory of Computing (STOC), pp. 121–129 (1993)

Papadimitriou, C.H.: Algorithms, games, and the internet. In: Proc. on 33rd Annual ACM Symposium on Theory of Computing (STOC), pp. 749–753. ACM, Heraklion (2001)

Papadimitriou, C.H.: On inefficient proofs of existence and complexity classes. In: Proceedings of the 4th Czechoslovakian Symposium on Combinatorics 1990, Prachatice (1991)

Papadimitriou, C.H.: On selecting a satisfying truth assignment. Proceedings 32nd Annual Symposium on Foundations of Computer Science, pp. 163–169. San Juan, Puerto Rico (1991)

Papadimitriou, C.H.: On the complexity of the parity argument and other inefficient proofs of existence. J. Comput. Syst. Sci. **48**, 498–532 (1994)

Papadimitriou, C.H.: The Euclidean travelling salesman problem is NP-complete. Theor. Comput. Sci. **4**, 237–244 (1977)

Papadimitriou, C.H.: The NP-completeness of the bandwidth minimization problem. Computing **16**(3), 263–270 (1976)

Papadimitriou, C.H., Schaffer, A., Yannakakis, M.: On the complexity of local search. In: 22nd Symp. on Theory of Computing (STOC), pp. 438 – 445 (1990)

Papadimitriou, C.H., Wolfe, D.: The complexity of facets resolved. J. Comput. Syst. Sci. **37**, 2–13 (1988)

Papadimitriou, C.H., Yannakakis, M.: On limited nondeterminism and the complexity of the V-C dimension. J. Comput. Syst. Sci. **53**(2), 161–170 (1996)

Papadimitriou, C.H., Yannakakis, M.: Shortest Paths without a Map. Theor. Comput. Sci. **84**, 127–150 (1991) Preliminary version in ICALP '89

Papadimitriu, C.H., Tsitsiklis, J.N.: The complexity of markov decision processes. In: Mathematics of Operations Research, 1987, pp. 441–450.

Papaefthymiou, M.C.: Asymptotically Efficient Retiming under Setup and Hold Constraints. In: Proceedings of the IEEE/ACM International Conference on Computer-Aided Design,pp. 288–295, November 1998

Park, K., Galil, Z.: Truly alphabet-independent two-dimensional pattern matching. In: Proceeding 33rd IEEE FOCS, 1992, pp. 247–256

Parkes, D.C.: Chapter 2: Iterative Combinatorial Auctions. Ph. D. thesis, University of Pennsylvania (2004)

Parra, A., Scheffler, P.: Characterizations and algorithmic applications of chordal graph embeddings. Discret. Appl. Math. **79**, 171–188 (1997)

Parvaresh, F., Vardy, A.: Correcting errors beyond the Guruswami—Sudan radius in polynomial time. In: Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science, pp. 285–294. Pittsburgh, 2005

Parvédy, P.R., Raynal, M.: Optimal Early Stopping Uniform Consensus in Synchronous Systems with Process Omission Failures. In: Proc. 16th Annual ACM Symposium on Parallel Algorithms (SPAA), pp. 302–310, Spain, June 2004

Pasco, R.: Source Coding Algorithms for Fast Data Compression, Ph. D. thesis, Stanford University (1976)

Pascoal, M.: Implementations and empirical comparison of k shortest loopless path algorithms. In: 9th DIMACS Implementation Challenge Workshop: Shortest Paths, DIMACS Center, Piscataway, NJ, 13–14 Nov 2006

Pătraşcu, M., Demain, E.D.: Lower Bounds for Dynamic Connectivity. In: Proc. 36th ACM Symposium on Theory of Computing (STOC), 2004, pp. 546–553

Pătraşcu, M., Demaine, E.: Logarithmic Lower Bounds in the Cell-Probe Model. SIAM J. Comput. **35**(4), 932–963 (2006) (presented at ACM STOC 2004)

Pătraşcu, M., Tarniţă, C.: On dynamic bit-probe complexity. Theor. Comput. Sci. **380**, 127–142 (2007). See also ICALP'05

Pătraşcu, M., Tarniţă, C.: On Dynamic Bit-Probe Complexity, Theoretical Computer Science, Special Issue on ICALP'05. In: Italiano, G.F., Palamidessi, C. (eds.) vol. 380, pp. 127–142 (2007) A preliminary version in Proc. 32nd International Colloquium on Automata, Languages and Programming (ICALP'05), 2005, pp. 969–981

Pătraşcu, M., Thorup, M.: Randomization does not help searching predecessors. In: Proc. 18th ACM/SIAM Symposium on Discrete Algorithms (SODA), 2007

Pătraşcu, M., Thorup, M.: Time-space trade-offs for predecessor search. In: Proc. 38th ACM Symposium on Theory of Computing (STOC), 2006, pp. 232–240

Patt-Shamir, B., Rajsbaum, S.: A theory of clock synchronization. In: Proceedings of the 26th Annual ACM Symposium on Theory of Computing, pp. 810–819, Montreal 23–25 May 1994

Pattengale, N.D., Moret, B.M.E.: A Sublinear-Time Randomized Approximation Scheme for the Robinson–Foulds Metric. In: Proceedings of the Tenth ACM Annual International Conference on Research in Computational Molecular Biology (RECOMB), pp. 221–230. Venice, Italy, April 2–5 2006

Paturi, R., Pudlák, P., Saks, M., Zane, F.: An Improved Exponential-time Algorithm for *k*-SAT. J. ACM **52**(3), 337–364 (2005) (An earlier version presented in Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science, 1998, pp. 628–637)

Paturi, R., Pudlák, P., Saks, M.E., Zane, F.: An improved exponential-time algorithm for *k*-SAT. Proceedings 39th Annual Symposium on Foundations of Computer Science, pp. 628–637. Palo Alto, USA (1998) Also, J. ACM **52**(3), 337–364 (2006)

Paturi, R., Pudlák, P., Zane, F.: Satisfiability Coding Lemma. In: Proceedings of the 38th Annual IEEE Symposium on Foundations of Computer Science, 1997, pp. 566–574. Chicago J. Theor. Comput. Sci. (1999), http://cjtcs.cs.uchicago.edu/

Paul, C., Proskurowski, A., Telle, J.A.: Generating graphs of bounded branchwidth. In: Proceedings of the 32nd Workshop on Graph Theoretic Concepts in Computer Science (WG 2006). Lecture Notes Computer Science, vol. 4271, pp. 205–216. Springer, Berlin (2006)

Paul, C., Telle, J.A.: New tools and simpler algorithms for branchwidth. In: Proceedings of the 13th Annual European Symposium on Algorithms (ESA 2005), 2005 pp. 379–390

Paul, J., Simon, W.: Decision trees and random access machines. In: Symposium über Logik und Algorithmik. (1980) See also Mehlhorn, K.: Sorting and Searching, pp. 85–97. Springer, Berlin (1984)

Pearl, J.: Capacity and error-estimates for boolean classifiers with limited complexity. IEEE Trans. on Pattern Recognition and Machine Intelligence, PAMI-**1**(4), 350–356 (1979)

Pearl, J.: Heuristics: Intelligent Search Strategies for Computer Problem Solving. Addison-Wesley, Reading, MA (1984)

Pearson, W.R., Lipman, D.J.: Improved Tools for Biological Sequence Comparison. Proc. Natl. Acad. Sci. USA **85**, 2444–2448 (1988)

Pease, M.C., Shostak, R.E., Lamport, L.: Reaching Agreement in the Presence of Faults. J. ACM **27**(2), 228–234 (1980)

Pelc, A., Peleg, D.: Feasibility and complexity of broadcasting with random transmission failures. Proc. 24th Ann. ACM Symposium on Principles of Distributed Computing (PODC), pp. 334–341, Las Vegas, July 17–20 2005

Peleg, D.: Distributed Computing: A Locality-Sensitive Approach. In: SIAM Monographs on Discrete Mathematics and Applications 5 (2000)

Peleg, D., Schäffer, A.A.: Graph spanners. J. Graph Theory **13**, 99–116 (1989)

Peleg, D., Ullman, J.D.: An optimal synchronizer for the hypercube. SIAM J. Comput. **18**, 740–747 (1989)

Peleg, D., Upfal, E.: A trade-off between space and efficiency for routing tables. J. Assoc. Comput Mach. **36**(3), 510–530 (1989)

Peleg, D., Wool, A.: Crumbling walls: A class of practical and efficient quorum systems. Distrib. Comput. **10**, 87–98 (1997)

Peleg, D., Wool, A.: The availability of quorum systems. Inf. Comput. **123**, 210–223 (1995)

Pemmaraju, S., Raman, R., Varadarajan, K.: Buffer minimization using max-coloring. In: Proc. of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2004), pp. 562–571. (2004)

Pennebaker, W.B., Mitchell, J.L., Langdon, G.G., Arps, R.B.: An overview of the basic principles of the Q-coder adaptive binary arithmetic coder. IBM J. Res. Develop. **32**, 717–726 (1988)

Penrose, M.: On *k*-connectivity for a geometric random graph. Random. Struct. Algorithms **15**(2), 145–164 (1999)

Penrose, M.: Random Geometric Graphs. Oxford University Press, Oxford (2003)

Penrose, M.: The longest edge of the random minimal spanning tree. Ann. Appl. Probab. **7**(2), 340–361 (1997)

Pereira, F., Riley, M.: Speech recognition by composition of weighted finite automata. In: Finite-State Language Processing, pp. 149–173. MIT Press, Cambridge (1997)

Perkins, C.E.: Ad Hoc Networking. Addison-Wesley, Boston (2001)

Perry, K.J., Toueg, S.: Distributed Agreement in the Presence of Processor and Communication Faults. IEEE Trans. Softw. Eng. **12**(3), 477–482 (1986)

Peters, J.G., Rudolph, L.: Parallel aproximation schemes for subset sum and knapsack problems. Acta Inform. **24**, 417–432 (1987)

Peterson, G.L.: Concurrent reading while writing. ACM Trans. Program. Lang. Syst. **5**(1), 56–65 (1983)

Peterson, G.L., Burns, J.E.: Concurrent reading while writing II: The multiwriter case. In: Proc. 28th IEEE Symp. Found. Comput. Sci., pp. 383–392. Los Angeles, 27–29 October 1987

Peterson, W.W.: Encoding and error-correction procedures for Bose-Chaudhuri codes. IEEE Trans. Inf. Theory. **6**, 459–470 (1960)

Pettie, S. Ramachandran, V.: An Optimal Minimum Spanning Tree Algorithm. J. ACM **49**(1), 16–34 (2002)

Pettie, S.: A new approach to all-pairs shortest paths on real-weighted graphs. Theor. Comput. Sci. **312**(1), 47–74 (2004)

Pettie, S.: Low-Distortion Spanners. In: 34th International Colloquium on Automata Languages and Programm, Wroclaw, July 2007, pp. 78–89

Pettie, S.: On the comparison-addition complexity of all-pairs shortest paths. In: Proc. 13th Int'l Symp. on Algorithms and Computation (ISAAC), 2002, pp. 32–43

Pettie, S.: On the shortest path and minimum spanning tree problems. Ph.D. thesis, The University of Texas, Austin, August 2003

Pettie, S.: Towards a final analysis of pairing heaps. In: Proc. 46th Annual Symposium on Foundations of Computer Science (FOCS), 2005, pp. 174–183

Pettie, S., Ramachandran, V.: A randomized time-work optimal parallel algorithm for finding a minimum spanning forest. SIAM J. Comput. **31**(6), 1879–1895 (2002)

Pettie, S., Ramachandran, V.: A shortest path algorithm for real-weighted undirected graphs. SIAM J. Comput. **34**(6), 1398–1431 (2005)

Pettie, S., Ramachandran, V.: Minimizing randomness in minimum spanning tree, parallel connectivity and set maxima algorithms. In: Proc. 13th ACM-SIAM Symp. on Discrete Algorithms (SODA), 2002, pp. 713–722

Pettie, S., Ramachandran, V.: New randomized minimum spanning tree algorithms using exponentially fewer random bits. ACM Trans. Algorithms. **4**(1), article 5 (2008)

Pettie, S., Ramachandran, V., Sridhar, S.: Experimental evaluation of a new shortest path algorithm. In: Proc. 4th Workshop on Algorithm Engineering and Experiments (ALENEX), 2002, pp. 126–142

Pevsner, J.: Bioinformatics and functional genomics. Wiley, New York (2003)

Pevtsov, S., Fedulova, I., Mirzaei, H., Buck, C., Zhang, X.: Performance evaluation of existing de novo sequencing algorithms. J. Proteome Res. **5**(11), 3018–3028 (2006) ASAP Article 10.1021/pr060222h

Pevzner, P., Tesler, G.: Human and mouse genomic sequences reveal extensive breakpoint reuse in mammalian evolution. PNAS **100**, 7672–7677 (2003)

Pevzner, P.A.: Computational molecular biology: an algorithmic approach. MIT Press, Cambridge, MA (2000)

Pevzner, P.A.: Multiple alignment, communication cost, and graph matching. SIAM J. Appl. Math. **52**, 1763–1779 (1992)

Pevzner, P.A., Sze, S.H.: Combinatorial approaches to finding subtle signals in DNA sequences. In: Proc. of 8th ISMB, pp. 269–278. AAAI Press, (2000)

Pinedo, M.: Scheduling: Theory, Algorithms and Systems, 2nd ed. Prentice Hall, Englewood Cliffs (2002)

Pion, S., Yap, C.: Constructive root bound method for k-ary rational input numbers, September, (2002). Extended Abstract. Submitted, (2003) ACM Symposium on Computational Geometry

Pitt, L.: Inductive inference, DFAs, and computational complexity. In: Analogical and Inductive Inference, 2nd International Workshop, Reinhardsbrunn Castle, GDR. Lecture Notes in Computer Science, vol. 397, pp. 18–44. Springer, Berlin (1989)

Pitt, L., Valiant, L.: Computational limitations on learning from examples. J. ACM **35**(4), 965–984 (1988)

Platt, J.: Fast training of support vector machines using sequential minimal optimization. In: Schölkopf, B., Burges, C.J.C., Smola, A.J. (eds.) Advances in Kernel Methods Support Vector Learning. pp 185–208. MIT Press, Cambridge (1999)

Plaxton, C., Rajaraman, R., Richa, A.: Accessing nearby copies of replicated objects in a distributed environment. In: Proceedings of the Ninth Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA 97), 1997, pp. 311–320

Plehn, J., Voigt, B.: Finding minimally weighted subgraphs. Lect. Notes Comput. Sci. **484**, 18–29 (1990)

Plotkin, S.A., Shmoys, D.B., Tardos, É.: Fast approximation algorithms for fractional packing and covering problems. In: Proceedings of 32nd Annual IEEE Symposium on Foundations of Computer Science (FOCS), 1991, pp. 495–504

Plotkin, S.A., Shmoys, D.B., Tardos, É.: Fast approximation algorithms for fractional packing and covering problems. Math. Oper. Res. **20**(2) 257–301 (1995). Preliminary version appeared in [6]

Podtelezhnikov, A., Cozzarelli, N., Vologodskii, A.: Equilibrium distributions of topological states in circular DNA: interplay of supercoiling and knotting. (English. English summary) Proc. Natl. Acad. Sci. USA **96**(23), 12974–129 (1999)

Polischuk, A., Spielman, D.: Nearly linear-size holographic proofs. In: Proceedings of the Twenty-Sixth Annual ACM Symposium on the Theory of Computing, pp. 194–203. ACM, New York (1994)

Poljak, S.: Polyhedral and eigenvalue approximations of the max-cut problem. Sets, Graphs and Numbers. Colloqiua Mathematica Societatis Janos Bolyai **60**, 569–581 (1992)

Poljak, S., Rendl, F.: Node and edge relaxations of the max-cut problem. Comput. **52**, 123–137 (1994)

Poljak, S., Rendl, F.: Nonpolyhedral relaxations of graph-bisection problems. SIAM J. Opt. **5**, 467–487 (1995)

Poljak, S., Rendl, F.: Solving the max-cut using eigenvalues. Discret. Appl. Math. **62**(1–3), 249–278 (1995)

Poljak, S., Tuza, Z.: Maximum cuts and large bipartite subgraphs. DIMACS Ser. Discret. Math. Theor. Comput. Sci. **20**, 181–244 (1995)

Pomerance, C.: Factoring. In: Pomerance, C. (ed.) Cryptology and Computational Number Theory, Proceedings of Symposia in Applied Mathematics, vol. 42, p. 27. American Mathematical Society

Poon, C.K., Ramachandran, V.: A randomized linear-work EREW PRAM algorithm to find a minimum spanning forest. Algorithmica **35**(3), 257–268 (2003)

Popper, K.: The Logic of Scientific Discovery. Harper & Row, New York (1959)

Posada, D., Crandall, K.A.: Intraspecific gene genealogies: trees grafting into networks. TRENDS Ecol. Evol. **16**(1), 37–45 (2001)

Potts, R.: Some generalized order - disorder transformations, Proc. Camb. Phil. Soc. **48**, 106–109 (1952)

Powell, O., Leone, P., Rolim, J.: Energy Optimal Data Propagation in Sensor Networks. J. Prarallel Distrib. Comput. **67**(3), 302–317 (2007) http://arxiv.org/abs/cs/0508052

Powell, O., Nikolesteas, S.: Simple and efficient geographic routing around obstacles for wireless sensor networks. In: WEA 6th Workshop on Experimental Algorithms, Rome, Italy. Springer, Berlin (2007)

Preparata, F.P., Shamos, M.I.: Computational Geometry: an Introduction. Springer, New York (1985)

Price, B., Baecker, R., Small, I.: A Principled Taxonomy of Software Visualization. J. Visual Lang. Comp. **4**, 211–266 (1993)

Prieto-Rodriguez, E.: Systematic kernelization in FPT algorithm design. Dissertation, School of Electrical Engineering and Computer Science, University of Newcastle, Australia (2005)

Prokop, H.: Cache-oblivious algorithms. Master's thesis, Massachusetts Institute of Technology, Dept. of Electrical Engineering and Computer Science (1999)

Proll, L.G.: A simple method of assigning projects to students. Oper. Res. Q. **23**(23), 195–201 (1972)

Pruhs, K., Sgall, J., Torng, E.: Online scheduling. In: Handbook on Scheduling: Algorithms, Models and Performance Analysis, CRC press (2004). Symposium on Theory of Computing (STOC), pp. 110–119. (1997)

Pruhs, K., Uthaisombut, P., Woeginger, G.: Getting the Best Response for Your Erg. In: Scandanavian Workshop on Algorithms and Theory, 2004

Przytycka, T.M.: Transforming rooted agreement into unrooted agreement. J. Comput. Biol. **5**(2), 335–349 (1998)

Pudlák, P.: Satisfiability – algorithms and logic. In: Proceedings of the 23rd International Symposium on Mathematical Foundations of Computer Science, MFCS'98. Lecture Notes in Computer Science, vol. 1450, pp. 129–141. Springer, Berlin (1998)

Pugh, W.: Skip lists: A probabilistic alternative to balanced trees. Commun. ACM **33**, 668–676 (1990)

Pugh, W.: Skip lists: A probabilistic alternative to balanced trees. In: Workshop on Algorithms and Data Structures, 1989, pp. 437–449

Puglisi, S., Smyth, W., Turpin, A.: A taxonomy of suffix array construction algorithms. ACM Comput. Surv. **39**(2), Article 4, 31 pages (2007)

Puterman, M.: Markov Decision Processes. Wiley-Interscience, New York (1994)

Pyrga, E., Schulz, F., Wagner, D., Zaroliagis, C.: Efficient Models for Timetable Information in Public Transportation Systems. ACM J. Exp. Algorithmic **12**(2.4), 1–39 (2007)

Pyrga, E., Schulz, F., Wagner, D., Zaroliagis, C.: Experimental comparison of shortest path approaches for timetable information. In: Proceedings 6th Workshop on Algorithm Engineering and Experiments (ALENEX), Society for Industrial and Applied Mathematics, 2004, pp. 88–99

Pyrga, E., Schulz, F., Wagner, D., Zaroliagis, C.: Towards realistic modeling of time-table information through the time-dependent approach. In: Proceedings of the 3rd Workshop on Algorithmic Methods and Models for Optimization of Railways (ATMOS'03), 2003, [1], pp. 85–103

Qiu, L., Padmanabhan, V.N., Voelker, G.: On the placement of web server replicas. In: Proceedings of the 20th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), pp. 1587–1596. IEEE Computer Society, Los Alamitos (2001)

Quine, W.V.: A way to simplify truth functions. Am. Math. Mon. **62**(9), 627–631 (1955)

Quine, W.V.: The problem of simplyfying truth functions. Am. Math. Mon. **59**(8), 521–531 (1952)

Rabani, Y., Tardos, E.: Distributed Packet Switching in Arbitrary Networks. In: the 28th ACM Symposium on Theory of Computing, pp. 366–376 (1996)

Rabin, M.: Randomized Byzantine Generals. In: Proc. 24th Annual ACM Symposium on Foundations of Computer Science, 1983, pp. 403–409

Rabin, M.O., Vazirani, V.V.: Maximum Matchings in General Graphs Through Randomization. J. Algorithms **10**, 557–567 (1989)

Rabinovich, Y., Raz, R.: Lower bounds on the distortion of embedding finite metric spaces in graphs. Discret. Comput. Geom. **19**, 79–94 (1998)

Räcke, H.: Minimizing congestion in general networks. In: Proceedings of the 43rd Annual Symposium on the Foundations of Computer Science, pp. 43–52 (2002)

Raghavachari, B.: Algorithms for finding low degree structures. In: Hochbaum, D.S. (ed.) Approximation Algorithms for NP-Hard Problems. pp. 266–295. PWS Publishing Company, Boston (1995)

Raghavan, P., Thompson, C.D.: Randomized rounding: a technique for provably good algorithms and algorithmic proofs. Combinatorica **7**, 365–374 (1987)

Rahman, N., Cole, R., Raman, R.: Optimised predecessor data structures for internal memory. In: Proc. Algorithm Engineering, 5th International Workshop, WAE. LNCS, vol. 2141, pp. 67–78. Springer, Berlin (2001)

Rahman, N., Raman, R.: Adapting radix sort to the memory hierarchy. ACM J. Exp. Algorithmics **6**, Article 7 (2001)

Rahman, N., Raman, R.: Analysing cache effects in distribution sorting. ACM J. Exp. Algorithmics **5**, Article 14 (2000)

Rahman, N., Raman, R.: Cache analysis of non-uniform distribution sorting algorithms. (2007) http://www.citebase.org/abstract?id=oai:arXiv.org:0706.2839 Accessed 13 August 2007 Preliminary version in: Proc. of 8th Annual European Symposium on Algorithms (ESA 2000). LNCS, vol. 1879, pp. 380–391. Springer, Berlin Heidelberg (2000)

Raipin Parvedy, P., Raynal, M., Travers, C.: Early-stopping *k*-set agreement in synchronous systems prone to any number of process crashes. In: Proc. 8th Int'l Conference on Parallel Computing Technologies (PaCT'05). LNCS, vol. 3606, pp. 49–58. Springer, Berlin (2005)

Raipin Parvedy, P., Raynal, M., Travers, C.: Strongly-terminating early-stopping *k*-set agreement in synchronous systems with general omission failures. In: Proc. 13th Colloquium on Structural Information and Communication Complexity (SIROCCO'06). LNCS, vol. 4056, pp. 182–196. Springer, Berlin (2006)

Rajagopalan, S., Vazirani, V.V.: On the bidirected cut relaxation for the metric Steiner tree problem. In: 10th ACM-SIAM Symposium on Discrete Algorithms, Baltimore, (1999), pp. 742–751

Rajan, V., Ghosh, R., Gupta, P.: An efficient parallel algorithm for random sampling. Inf. Process. Lett. **30**, 265–268 (1989)

Rajaraman, R., Wong, D.F.: Optimum clustering for delay minimization. IEEE Trans. Comput.-Aided Des. Integr. Circ. Syst. **14**, 1490–1495 (1995)

Rajkumar, R.: Synchronization In Real-Time Systems – A Priority Inheritance Approach. Kluwer Academic Publishers, Boston (1991)

Ramalingam, G.: Bounded incremental computation. In: Lecture Notes in Computer Science, vol. 1089. Springer, New York (1996)

Ramalingam, G.: Bounded incremental computation. Lect. Note Comp. Sci. 1089 (1996)

Ramalingam, G., Reps, T.: An incremental algorithm for a generalization of the shortest path problem. J. Algorithm **21**, 267–305 (1996)

Ramalingam, G., Reps, T.: On the computational complexity of dynamic graph problems. Theor. Comp. Sci. **158**, 233–277 (1996)

Raman, R., Raman, V., Rao, S.S.: Succinct indexable dictionaries with applications to encoding *k*-ary trees and multisets. In: Proc. 13th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 233–242. San Francisco, USA (2002)

Raman, R., Rao, S. S.: Succinct dynamic dictionaries and trees. In: Baeten, J.C.M., Lenstra, J.K., Parrow J., Woeginger, G.J. (eds.) Proceedings of the 30th International Colloquium on Automata, Languages and Programming, pp. 357–368. Springer, Heidelberg (2003)

Ramanathan, S., Loyd, E.R.: The Complexity of Distance 2-Coloring. In: Proceedings of the 4th International Conference of Computing and Information, pp. 71–74 (1992)

Ramasubramanian, V., Sirer, E.G.: Beehive: O(1) lookup performance for power-law query distributions in peer-to-peer overlays. In: Proceedings of Networked System Design and Implementation (NSDI), 2004

Ramasubramanian, V., Sirer, E.G.: The design and implementation of a next generation name service for the internet. In: Proceedings of SIGCOMM, 2004

Randerath, B., Schiermeyer, I.: Exact algorithms for MINIMUM DOMINATING SET. Technical Report, zaik-469, Zentrum für Angewandte Informatik Köln (2004)

Ranjan, R., Aziz, A., Brayton, R., Plessier, B., Pixley, C.: Efficient BDD Algorithms for FSM Synthesis and Verification. In: Proceedings of the International Workshop on Logic Synthesis, May 1995

Rao, S.: Small distortion and volume preserving embeddings for planar and Euclidean metrics. In: Proceedings of the 15th Annual Symposium on Computational Geometry, pp. 300–306. ACM, New York (1999)

Rao, S., Smith, W.D.: Approximating geometrical graphs via spanners and banyans. In: Proceedings of the 30th ACM Symposium on Theory of Computing, pp. 540–550. Dallas, 23–26 May 1998

Rao, S.B.: Faster algorithms for finding small edge cuts in planar graphs (extended abstract). In: Proceedings of the Twenty-Fourth Annual ACM Symposium on the Theory of Computing, pp. 229–240, May (1992)

Rappaport, T.S.: Wireless Communications: Principles and Practices. Prentice Hall, IEEE Press, Piscataway (1996)

Raptopoulos, C., Spirakis, P.: Simple and efficient greedy algorithms for hamiltonian cycles in random intersection graphs. In: Proceedings of the 16th International Symposium on Algorithms and Computation (ISAAC), pp 493–504. Springer, Berlin Heidelberg (2005)

Rastegari, B., Condon, A.: Linear time algorithm for parsing RNA secondary structure. In: Casadio, R., Myers, E.: (eds.) Proc. 5th Workshop Algs. in Bioinformatics (WABI'05). Lecture Notes in Computer Science, vol. 3692, pp. 341–352. Springer, Mallorca, Spain (2005)

Rastegari, B., Condon, A.: Parsing nucleic acid pseudoknotted secondary structure: algorithm and applications. J. Comput. Biol. **14**(1), 16–32 (2007)

Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A scalable content-addressable network. In: Proceedings of the ACM SIGCOMM 2001 Technical Conference, 2001

Raubeson, L.A., Jansen, R.K.: Chloroplast DNA evidence on the ancient evolutionary split in vascular land plants. Science **255**, 1697–1699 (1992)

Ravi, R., Singh, M.: Delegate and conquer: An LP-based approximation algorithm for minimum degree MSTs. In: Proceedings of the 33rd International Colloquium on Automata, Languages and Programming (ICALP 2006) Part I. LNCS, vol. 4051, pp. 169–180. Springer, Berlin (2006)

Ravi, R., Sinha, A.: Hedging uncertainty: Approximation algorithms for stochastic optimization problems. Math. Program. **108**(1), 97–114 (2006)

Ravi, R., Sinha, A.: Integrated logistics: Approximation algorithms combining facility location and network design. In: Proceedings of the 9th Conference on Integer Programming and Combinatorial Optimization (IPCO). Lecture Notes in Computer Science, vol. 2337, pp. 212–229. Springer, Berlin (2002)

Ravi, R., Sinha, A.: Multicommodity facility location. In: Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 342–349. SIAM, Philadelphia (2004)

Ravishankar, K., Singh, S.: Broadcasting on $[0, L]$. Discret. Appl. Math. **53**, 299–319 (1994)

Raynal, M.: Algorithms for mutual exclusion. MIT Press, Cambridge (1986). Translation of: Algorithmique du parallélisme, (1984)

Raynal, M.: Consensus in Synchronous Systems: A Concise Guided Tour. In: Proc. 9th Pacific Rim International Symposium on Dependable Computing (PRDC), pp. 221–228, Japan, December 2002

Raynal, M., Singhal, M.: Capturing causality in distributed systems. IEEE Comput. **29**, 49–56 (1996)

Raynal, M., Travers, C.: Synchronous set agreement: a concise guided tour (including a new algorithm and a list of open problems). In: Proc. 12th Int'l IEEE Pacific Rim Dependable Computing Symposium (PRDC'2006), pp. 267–274. IEEE Society Computer Press, Los Alamitos (2006)

Raz, R., Safra, S.: A sub-constant error-probability low-degree test, and a sub-constant error-probability pcp characterization of np. In: Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, pp. 475–484. ACM, New York (1997)

Razborov, A.: On the Distributional Complexity of Disjointness. Theor. Comput. Sci. **106**(2), 385–390 (1992)

Razborov, A.A.: Quantum communication complexity of symmetric predicates. Izvestiya of the Russian Academy of Science, Mathematics, **67**, 145–159 (2002)

Reda, S., Chowdhary, A.: Effective linear programming based placement methods. In: ACM Press, San Jose, 9–12 Apr 2006

Reed, B., Smith, K., Vetta, A.: Finding odd cycle transversals. Oper. Res. Lett. **32**(4), 299–301 (2004)

Reed, B.A.: Algorithmic aspects of tree width, pp. 85–107. CMS Books Math. Ouvrages Math. SMC, 11. Springer, New York (2003)

Reed, B.A.: Tree width and tangles, a new measure of connectivity and some applications, LMS Lecture Note Series, vol. 241, pp. 87–162. Cambridge University Press, Cambridge (1997)

Reed, D.A., Aydt, R.A., Noe, R.J., Roth, P.C., Shields, K.A., Schwartz, B., Tavera, L.F.: Scalable performance analysis: The Pablo performance analysis environment. In: Skjellum, A., (ed) Proc. Scalable Parallel Libraries Conf., pp. 104–113, Mississippi State University, October 1993. IEEE Computer Society Press

Reeder, J., Giegerich, R.: Design, implementation and evaluation of a practical pseudoknot folding algorithm based on thermodynamics. BMC Bioinform. **5**, 104 (2004)

Regev, O.: New Lattice-Based Cryptographic Constructions. J. ACM **51**, 899–942 (2004)

Régnier, M., Rostami, L.: A unifying look at d-dimensional periodicities and space coverings. In: 4th Symp. on Combinatorial Pattern Matching, 15, 1993

Reichardt, B.W.: Error-detection-based quantum fault tolerance against discrete Pauli noise. Ph. D. thesis, University of California, Berkeley (2006). quant-ph/0612004

Reichardt, B.W., Grover, L.K.: Quantum error correction of systematic errors using a quantum search framework. Phys. Rev. A **72**, 042326 (2005)

Reingold, N., Westbrook, J., Sleator, D.D.: Randomized Competitive Algorithms for the List Update Problem. Algorithmica **11**(1), 15–32 (1994) (Conference version included author Irani, S.: SODA 1991, pp. 251–260)

Remy, J., Steger, A.: A Quasi-Polynomial Time Approximation Scheme for Minimum Weight Triangulation. In: Proceedings 38th ACM Symposium on Theory of Computing (STOC'06). ACM Press, New York, NY, USA (2006)

Ren, J., Rastegart, B., Condon, A., Hoos, H.: HotKnots: Heuristic prediction of rna secondary structure including pseudoknots. RNA **11**, 1194–1504 (2005)

Renner, R.: Security of Quantum Key Distribution. Ph. D. thesis, Swiss Federal Institute of Technology (ETH) Zurich, Also available at http://arxiv.org/abs/quant-ph/0512258 (2005)

Renner, R., König, R.: Universally composable privacy amplification against quantum adversaries. In: Second Theory of Cryptography Conference TCC. Lecture Notes in Computer Science, vol. 3378, pp. 407–425. Springer, Berlin (2005). Also available at http://arxiv.org/abs/quant-ph/0403133

Reussner, R., Sanders, P., Träff, J.: SKaMPI: A comprehensive benchmark for public benchmarking of MPI. Scientific Programming, 2001. accepted, conference version with Prechelt, L., Müller, M. In: Proc. EuroPVM/MPI (1998)

Reyzin, L., Srivastava, N.: On the longest path algorithm for reconstructing trees from distance matrices. Inf. Process. Lett. **101**, 98–100 (2007)

Rhea, S., Geels, D., Roscoe, T., Kubiatowicz, J.: Handling churn in a dht. Tech. Report Technical Report UCB//CSD-03-1299, The University of California, Berkeley, December 2003

Riany, Y., Shavit, N., Touitou, D.: Towards a practical snapshot algorithm. Theor. Comput. Sci. **269**, 163–201 (2001)

Richardson, D.: How to recognize zero. J. Symb. Comput. **24**, 627–645 (1997)

Richter, P.C.: Quantum speedup of classical mixing processes. Phys. Rev. A **76**, 042306 (2007)

Riess, B.M., Doll, K., Frank, M.J.: Partitioning Very Large Circuits Using Analytical Placement Techniques. In: Proc. 31th ACM/IEEE Design Automation Conf., 1994, pp. 646–651

Rissanen, J.: Modeling by Shortest Data Description. Automatica **14**, 465–471 (1978)

Rivas, E., Eddy, S.: A dynamic programming algorithm for RNA structure prediction including pseudoknots. J. Mol. Biol. **285**, 2053–2068 (1999)

Rivas, E., Eddy, S.R.: Secondary structure alone is generally not statistically significant for the detection of noncoding RNAs. Bioinformatics **16**, 583–605 (2000)

Rivest, R.: On self-organizing sequential search heuristics. Commun. ACM **19**, 63–67 (1976)

Robert, J., Schabanel, N.: Non-Clairvoyant Batch Sets Scheduling: Fairness is Fair enough. Personal Correspondence (2007)

Roberts, K.: The characterization of implementable choice rules. In: Laffont, J.J. (ed.) Aggregation and Revelation of Preferences, pp. 321–349. North-Holland (1979)

Robertson, N. Seymour, P.D.: Graph minors. X. Obstructions to tree-decomposition J. Combin. Theor. Ser. B **52**, 153–190 (1991)

Robertson, N. Seymour, P.D.: Graph minors. XII. Distance on a surface. J. Combin. Theor. Ser. B **64**, 240–272 (1995)

Robertson, N., Seymour, P.: Graph minors. II. Algorithmic aspects of tree-width. J. Algorithms **7**, 309–322 (1986)

Robertson, N., Seymour, P.D.: Graph Minors XIII.The Disjoint Paths Problem. J. Comb. Theor. B **63**(1), 65–110 (1995)

Robertson, N., Seymour, P.D., Thomas, R.: Quickly excluding a planar graph. J. Combin. Theor. Ser. B **62**, 323–348 (1994)

Robin, G., Zelikovsky, A.: Improved Steiner trees approximation in graphs. In: SIAM-ACM Symposium on Discrete Algorithms (SODA), San Francisco, CA, pp. 770–779. January (2000)

Robins, G., Salowe, J.S.: Low-degree minimum spanning tree. Discret. Comput. Geom. **14**, 151–165 (1995)

Robinson, D.F.: Comparison of Labeled Trees with Valency Three. J. Comb. Theor. **11**, 105–119 (1971)

Robinson, D.F., Foulds, L.R.: Comparison of Phylogenetic Trees. Math. Biosci. **53**, 131–147 (1981)

Rodeh, M., Pratt, V., Even, S.: Linear algorithm for data compression via string matching. J. Assoc. Comput. Mach. **28**(1), 16–24 (1981)

Rodionov, V.: The parametric problem of shortest distances. USSR Comp. Math. Math. Phys. **8**(5), 336–343 (1968)

Roditty, L.: A faster and simpler fully dynamic transitive closure. In: Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms. ACM IEEE SODA, pp. 404–412. ACM, Baltimore (2003)

Roditty, L., Zwick, U.: A fully dynamic reachability algorithm for directed graphs with an almost linear update time. In: Proceedings of the 36th ACM Symposium on Theory of Computing. ACM STOC, pp. 184–191 ACM, Chicago (2004)

Roditty, L., Zwick, U.: Dynamic approximate all-pairs shortest paths in undirected graphs. In: Proc. of Symp. on Foundations of Computer Science, Rome, Oct. 2004, pp. 499–508

Roditty, L., Zwick, U.: Improved dynamic reachability algorithms for directed graphs. In: Proceedings of 43th Annual IEEE Symposium on Foundations of Computer Science (FOCS), Vancouver (2002), pp. 679–688

Roditty, L., Zwick, U.: On Dynamic Shortest Paths Problems. In: Proceedings of the 12th Annual European Symposium on Algorithms (ESA), Bergen (2004), pp. 580–591

Rødland, E.A.: Pseudoknots in RNA secondary structure: Representation, enumeration, and prevalence. J. Comput. Biol. **13**, 1197–1213 (2006)

Rodrigues, R., Liskov, B.: Rosebud: A scalable byzantine-fault tolerant storage architecture. In: Proceedings of the 18th ACM Symposium on Operating System Principles, San Francisco, USA (2003)

Rogers, H.: Theory of Recursive Functions and Effective Computability. McGraw-Hill, New York (1967)

Rohe, A.: Sequential and Parallel Algorithms for Local Routing. Ph. D. thesis, Bonn University, Bonn, Germany, Dec. (2001)

Rohnert, H.: A dynamization of the all-pairs least cost problem. In: Proc. 2nd Annual Symposium on Theoretical Aspects of Computer Science, (STACS'85). LNCS, vol. 182, pp. 279–286. Springer, Berlin (1985)

Röhrig, H.: Tree decomposition: A feasibility study. Master's thesis, Max-Planck-Institut für Informatik, Saarbrücken, Germany (1998)

Rolf, D.: 3-SAT $\in$ $RTIME(1.32971^n)$. In: ECCC TR03-054, 2003

Rolf, D.: 3-SAT $\in$ $RTIME(O(1.32793^n))$. ECCC TR03–054. (2003)

Rolf, D.: Improved Bound for the PPSZ/Schöning-Algorithm for 3-SAT. J. Satisf. Boolean Model. Comput. **1**, 111–122 (2006)

Roman, G., Cox, K.: A Declarative Approach to Visualizing Concurrent Computations. Computer **22**, 25–36 (1989)

Roman, G., Cox, K.: A Taxonomy of Program Visualization Systems. Computer **26**, 11–24 (1993)

Roman, G., Cox, K., Wilcox, C., Plun, J.: PAVANE: a System for Declarative Visualization of Concurrent Computations. J. Visual Lang. Comp. **3**, 161–193 (1992)

Romani, F.: Shortest-path problem is not harder than matrix multiplications. Info. Proc. Lett. **11**, 134–136 (1980)

Romero-Medina, A.: Implementation of stable solutions in a restricted matching market. Rev. Economic. Des. **3**(2), 137–147 (1998)

Ronen, A.: On approximating optimal auctions (extended abstract). In: Proc. 3rd ACM Conference on Electronic Commerce (EC), pp. 11–17 (2001)

Ronen, A., Saberi, A.: On the hardness of optimal auctions. In: Proc. 43rd Ann. IEEE Symp. on Foundations of Comput. Sci. (FOCS), pp. 396–405 (2002)

Ronn, E.: NP-complete stable matching problems. J. Algorithms **11**, 285–304 (1990)

Rose, D., Tarjan, R.E., Lueker, G.: Algorithmic aspects of vertex elimination on graphs. SIAM J. Comput. **5**, 146–160 (1976)

Rosenberg, A.L., Heath, L.S.: Graph separators, with applications. Frontiers of Computer Science. Kluwer Academic/Plenum Publishers, New York (2001)

Rosenblatt, F.: The perceptron: A probabilistic model for information storage and organization in the brain. Psychol. Rev. **65**, 386–407 (1958)

Rosenthal, R.W.: A class of games possessing pure-strategy Nash equilibria. Int. J. Game Theor. **2**, 65–67 (1973)

Ross, S.: Stochastic Processes. Wiley (1995)

Roth, A., Sönmez, T., Ünver, U.: Kidney Exchange. Quarter. J. Econ. **119**, 457–488 (2004)

Roth, A., Sotomayor, M.: Two-Sided Matching. Cambridge University Press, Cambridge (1990)

Roth, A.E.: The evolution of the labor market for medical interns and residents: a case study in game theory. J. Polit. Econ. **92**(6), 991–1016 (1984)

Roth, A.E., Postlewaite, A.: Weak versus strong domination in a market with indivisible goods. J. Math. Econ. **4**, 131–137 (1977)

Roth, A.E., Sotomayor, M.A.O.: Two-sided matching: a study in game-theoretic modeling and analysis. Econometric Society Monographs, vol. 18. Cambridge University Press, Cambridge, UK (1990)

Roth, A.E., Vande Vate, J.H.: Random paths to stability in two-sided matching. Econometrica **58**(6), 1475–1480 (1990)

Roth-Korostensky, C.: Algorithms for building multiple sequence alignments and evolutionary trees. Ph. D. Thesis, ETH Zürich, Institute of Scientific Computing (2000)

Rothkopf, M.: Scheduling with Random Service Times. Manag. Sci. **12**, 707–713 (1966)

Roughgarden, T.: Designing networks for selfish users is hard. In: 42nd IEEE Annual Symposium of Foundations of Computer Science, pp. 472–481 (2001)

Roughgarden, T.: Selfish Routing and the Price of Anarchy. The MIT Press, Cambridge (2005)

Roughgarden, T.: Selfish Routing. Dissertation, Cornell University, USA, May 2002, http://theory.stanford.edu/~tim/

Roughgarden, T.: Stackelberg scheduling strategies. In: 33rd ACM Annual Symposium on Theory of Computing, pp. 104–113 (2001)

Roughgarden, T., Tardos, E.: Bounding the inefficiency of equilibria in nonatomic congestion games. Games Econ. Behav. **47**, 389–403 (2004)

Roughgarden, T., Tardos, E.: How bad is selfish routing? In: 41st IEEE Annual Symposium of Foundations of Computer Science, pp. 93–102. J. ACM 49(2), pp 236–259, 2002, ACM, New York (2000)

Rowstron, A., Druschel, P.: Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In: IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), 2001, pp. 329–350

Roy, J.A., Adya, S.N., Papa, D.A., Markov, I.L.: Min-cut floorplacement. IEEE Trans. CAD **25**(7), 1313–1326 (2006)

Ruan, J., Stormo, G., Zhang, W.: An iterated loop matching approach to the prediction of RNA secondary structures with pseudoknots. Bioinformatics **20**, 58–66 (2004)

Ruan, L., Du, H., Jia, X., Wu, W., Li, Y., Ko, K.-I.: A greedy approximation for minimum connected dominating set. Theor. Comput. Sci. **329**, 325–330 (2004)

Ruan, L., Wu, W.: Broadcast routing with minimum wavelength conversion in WDM optical networks. J. Comb. Optim. **9** 223–235 (2005)

Rubinfeld, R.: On the robustness of functional equations. SIAM J. Comput. **28**(6), 1972–1997 (1999)

Rubinfeld, R., Sudan, M.: Robust characterization of polynomials with applications to program testing. SIAM J. Comput. **25**(2), 252–271 (1996)

Rubinstein, A.: Ranking the participants in a tournament. SIAM J. Appl. Math. **38**(1), 108–111 (1980)

Rudell, R.: Logic Synthesis for VLSI Design. Ph. D. thesis, University of California at Berkeley, ERL Memo 89/49, April 1989

Rudin III, J.F.: Improved bounds for the online scheduling problem. Ph. D. thesis, The University of Texas at Dallas (2001)

Rudin III, J.F., Chandrasekaran, R.: Improved bounds for the online scheduling problem. SIAM J. Comput. **32**, 717–735 (2003)

Ruf, N., Schöbel, A.: Set covering with almost consecutive ones property. Discret. Optim. **1**(2), 215–228 (2004)

Ruszinkó, M. On the upper bound of the size of *r*-cover-free families. J. Comb. Theory, Ser. A **66**, 302–310 (1984)

Ruzzo, W.L., Tompa, M.: A linear time algorithm for finding all maximal scoring subsequences. Proceedings of the 7th International Conference on Intelligent Systems for Molecular Biology, pp. 234–241 (1999)

Rytter, W.: Application of Lempel–Ziv factorization to the approximation of grammar-based compression. Theor. Comput. Sci. **302**(1–3), 211–222 (2003)

Rytter, W.: On maximal suffixes and constant-space linear-time versions of KMP algorithm. Theor. Comput. Sci. **299**(1–3), 763–774 (2003)

Rytter, W.: The Number of Runs in a String: Improved Analysis of the Linear Upper Bound. In: Proceedings of the 23rd Annual Symposium on Theoretical Aspects of Computer Science. Lecture Notes in Computer Science, vol. 3884, pp. 184–195. Springer, Berlin (2006)

Rytter, W.: The structure of subword graphs and suffix trees of Fibonacci words. In: Implementation and Application of Automata, CIAA 2005. Lecture Notes in Computer Science, vol. 3845, pp. 250–261. Springer, Berlin (2006)

Sadakane, K.: Compressed suffix trees with full functionality. Theor. Comput. Syst. **41**, 589–607 (2007)

Sadakane, K.: Compressed suffix trees with full functionality. Theory Comput. Syst. (2007) Online first. http://dx.doi.org/10.1007/s00224-006-1198-x

Sadakane, K.: New text indexing functionalities of the compressed suffix arrays. J. Algorithms **48**(2), 294–313 (2003)

Sadakane, K., Grossi, R.: Squeezing succinct data structures into entropy bounds. In: Proc. 17th ACM-SIAM SODA, pp. 1230–1239. ACM Press (2006)

Sagot, M.F.: Spelling approximate repeated or common motifs using a suffix tree. In: Proc. of the 3rd LATIN, vol. 1380 in LNCS, pp. 111–127. Springer (1998)

Şahinalp, C., Rajpoot, N.: Dictionary-based data compression: An algorithmic perspective. In: Sayood, K. (ed.) Lossless Compression Handbook, pp. 153–167. Academic Press, USA (2003)

Sahinalp, S.C., Vishkin, U.: Efficient approximate and dynamic matching of patterns using a labeling paradigm. In: Proc. of the Foundations of Computer Science (FOCS), 1996, pp. 320–328

Sahinalp, S.C., Vishkin, U.: Symmetry breaking for suffix tree construction. ACM STOC 300–309 (1994)

Sahni, S., Gonzalez, T.: P-complete approximation problems. J. ACM **23**(3), 555–565 (1976)

Saitou, N., Nei, M.: The neighbor-joining method: A new method for reconstruction of phylogenetic trees. Mol. Biol. Evol. **4**, 406–425 (1987)

Sakanushi, K., Kajitani, Y., Mehta, D.: The quarter-state-sequence floorplan representation. In: IEEE TCAS-I: **50**(3), 376–386 (2003)

Saks, M., Wigderson, A.: Probabilistic Boolean decision trees and the complexity of evaluating game trees. In: Proc. of 27th IEEE Symp. on Foundation of Computer Science (FOCS), Toronto, 27–29 October, pp. 29–38 (1986)

Saks, M., Yu, L.: Weak monotonicity suffices for truthfulness on convex domains. In: Proc. 6th ACM Conference on Electronic Commerce (ACM-EC), 2005, pp. 286–293

Saks, M., Zaharoglou, F.: Wait-free *k*-set agreement is impossible: The topology of public knowledge. In: Proceedings of the 25th

ACM Symposium on Theory of Computing, pp. 101–110, ACM Press, May 1993

Saks, M., Zaharoglou, F.: Wait-Free *k*-Set Agreement is Impossible: The Topology of Public Knowledge. SIAM J. Comput. **29**(5), 1449–1483 (2000)

Salomon, D.: Data Compression: the Complete Reference, 3rd edn. Springer, New York (2004)

Salomon, D.: Data Compression: the Complete Reference, 4th edn. Springer, London (2007)

Salowe, J.D.: Construction of multidimensional spanner graphs, with application to minimum spanning trees. In: ACM Symposium on Computational Geometry, 1991, pp. 256–261

Salowe, J.S.: Constructing multidimensional spanner graphs. Int. J. Comput. Geom. Appl. **1**(2), 99–107 (1991)

Samer, M., Szeider, S.: Algorithms for propositional model counting. In: Proceedings of LPAR 2007, 14th International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Yerevan, Armenia, 15–19 October 2007. Lecture Notes in Computer Science, vol. 4790, pp. 484–498. Springer, Berlin (2007)

Samorodnitsky, A., Trevisan, L.: A PCP characterization of NP with optimal amortized query complexity. In: Proceedings of the Thirty-Second Annual ACM Symposium on the Theory of Computing, pp. 191–199. ACM, New York (2000)

Samorodnitsky, A., Trevisan, L.: Gowers uniformity, influence of variables, and pcps. In: Thirty-Eighth ACM Symposium on Theory of Computing, pp. 11–20. ACM, New York (2006)

Sampathkumar, E., Walikar, H.B.: The Connected Domination Number of a Graph. J. Math. Phys. Sci. **13**, 607–613 (1979)

Sanchis, L.A.: Multiway Network Partitioning. IEEE Trans. Comput. **38**(1), 62–81 (1989)

Sanders, P.: Fast priority queues for cached memory. ACM J. Exp. Algorithmics **5**, Article 7 (2000)

Sanders, P., Schultes, D.: Engineering fast route planning algorithms. In: 6th Workshop on Experimental Algorithms. LNCS, vol. 4525, pp. 23–36. Springer, Berlin (2007)

Sanders, P., Schultes, D.: Engineering highway hierarchies. In: 14th European Symposium on Algorithms. LNCS, vol. 4168, pp. 804–816. Springer, Berlin (2006)

Sanders, P., Schultes, D.: Engineering Highway Hierarchies. In: Algorithms – ESA 2006. Lect. Note Comp. Sci. **4168**, 804–816 (2006)

Sanders, P., Schultes, D.: Highway hierarchies hasten exact shortest path queries. In: 13th European Symposium on Algorithms. LNCS, vol. 3669, pp. 568–579. Springer, Berlin (2005)

Sanders, P., Schultes, D.: Highway Hierarchies Hasten Exact Shortest Path Queries. In: Algorithms – ESA 2005. Lect. Note Comp. Sci. **3669**, 568–579 (2005)

Sanders, P., Schultes, D.: Robust, almost constant time shortest-path queries in road networks. In: 9th DIMACS Implementation Challenge Workshop: Shortest Paths, DIMACS Center, Piscataway, NJ, 13–14 Nov 2006

Sanderson, M.J., Purvis, A., Henze, C.: Phylogenetic supertrees: assembling the trees of life. TRENDS in Ecology & Evolution, **13**(3), 105–109 (1998)

Sankoff, D., Blanchette, M.: Multiple genome rearrangement and breakpoint phylogeny. J. Comp. Biol. **5**, 555–570 (1998)

Sankoff, D., Kruskal, J.B.: Time Warps, Strings Edits, and Macromolecules: The Theory and Practice of Sequence Comparison. Addison-Wesley (1983)

Sankoff, D., Rousseau, P.: Locating the vertices of a Steiner tree in arbitrary metric space. Math. Program. **9**, 240–246 (1975)

Sankoff, D.D.: Minimal mutation trees of sequences. SIAM J. Appl. Math. **28**, 35–42 (1975)

Sankowski, P.: Dynamic transitive closure via dynamic matrix inverse. In: FOCS '04: Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS'04), pp. 509–517. IEEE Computer Society, Washington, DC (2004)

Sankowski, P.: Processor Efficient Parallel Matching. In: Proceeding of the 17th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), 2005, pp. 165–170

Sankowski, P.: Subquadratic algorithm for dynamic shortest distances. In: 11th Annual International Conference on Computing and Combinatorics (COCOON'05), Kunming (2005), pp. 461–470

Sankowski, P.: Weighted Bipartite Matching in Matrix Multiplication Time. In: Proceedings of the 33rd International Colloquium on Automata, Languages and Programming, 2006, pp. 274–285

Santha, M.: On the Monte Carlo decision tree complexity of read-once formulae. Random Struct. Algorit. **6**(1), 75–87 (1995)

Santos, J.: K shortest path algorithms. In: 9th DIMACS Implementation Challenge Workshop: Shortest Paths, DIMACS Center, Piscataway, NJ, 13–14 Nov 2006

Sapatnekar, S.S.: RC interconnect optimization under the Elmore delay model. In: Proc. ACM/IEEE Design Automation Conf., pp. 387–391. ACM, New York (1994)

Sapatnekar, S.S., Deokar, R.B.: Utilizing the retiming-skew equivalence in a practical algorithm for retiming large circuits. IEEE Trans. Comput. Aided Des. **15**, 1237–1248 (1996)

Sapatnekar, S.S., Rao, V.B., Vaidya, P.M., Kang, S.M.: An Exact Solution to the Transistor Sizing Problem for CMOS Circuits Using Convex Optimization. IEEE Trans. Comput. Aided. Des. **12**(11), 1621–1634 (1993)

Sarkar, S., Tassiulas, L.: Fair distributed congestion control in multirate multicast networks. IEEE/ACM Trans. Netw. **13**(1), 121–133 (2005)

Savani, R., von Stengel, B.: Exponentially many steps for finding a nash equilibrium in a bimatrix game. In: Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS'04), pp. 258–267. Rome, 17–19 October 2004

Savari, S.: Redundancy of the Lempel–Ziv incremental parsing rule. IEEE Trans. Inf. Theor. **43**, 9–21 (1997)

Savoj, H.: Don't Cares in Multi-Level Network Optimization. Ph. D. thesis, University of California, Berkeley, Electronics Research Laboratory, College of Engineering. University of California, Berkeley, CA (1992)

Sawchuk, C.: Mobile Agent Rendezvous in the Ring. Ph. D. thesis, Carleton University, Ottawa, Canada (2004)

Schaetz T., Barrett, M.D., Leibfried, D., Chiaverini, J., Britton, J., Itano, W.M., Jost, J.D., Langer, C., Wineland, D.J.: Quantum Dense Coding with Atomic Qubits. Phys. Rev. Lett. **93**, 040505 (2004)

Schaffer, A. Yannakakis, M.: Simple local search problems that are hard to solve. SIAM J. Comput. **20**(1), 56–87 (1991)

Schapire, R.: The strength of weak learnability. Mach. Learn. **5**(2), 197–227 (1990)

Schapire, R.E., Sellie, L.M.: Learning sparse multivariate polynomials over a field with queries and counterexamples. J. Comput. Syst. Sci. **52**(2), 201–213 (1996)

Scheideler, C.: Universal Routing Strategies for Interconnection Networks. In: Lecture Notes in Computer Science, vol. 1390. Springer (1998)

Scherer, W., Scott, M.: Advanced contention management for dynamic software transactional memory. In: Proc. 24th Annual ACM Symposium on Principles of Distributed Computing, 2005

Schieber, B., Moran, S.: Slowing sequential algorithms for obtaining fast distributed and parallel algorithms: Maximum matchings. In: Proc. of 5th ACM Symp. on Principles of Distributed Computing, Calgary, 11–13 Aug. 1986, pp. 282–292

Schmeidler, D.: The Nucleolus of a Characteristic Function Game. SIAM J. Appl. Math. **17**, 1163–1170 (1969)

Schmidt, A., Vollmer, U.: Polynomial time quantum algorithm for the computation of the unit group of a number field. In: Proceedings of the 37th ACM Symposium on Theory of Computing. (2005)

Schmidt, J.P., Siegel, A.: The spatial complexity of oblivious *k*-probe hash functions. SIAM J. Comput. **19**(5), 775–786 (1990)

Schmidt, M.: Packet buffering: randomization beats deterministic algorithms. In: Proc. 22nd Annual Symp. on Theoretical Aspects of Computer Science (STACS). LNCS, vol. 3404, 293–304 (2005)

Schneider, F.B.: Implementing fault-tolerant services using the state machine approach: a tutorial. ACM Comput. Surv. **22**, 299–319 (1990)

Schneider, F.B.: Replication Management using the State-Machine Approach. In Sape Mullender, editor, Distributed Systems, pp. 169–197. ACM Press (1993)

Schnorr, C.P.: Fast LLL-type lattice reduction. Inform. Comput. **204**(1), 1–25 (2006)

Scholkopf, B., Smola, A.J.: Learning with Kernels. MIT Press, Cambridge (2002)

Schönhage, A., Strassen, V.: Schnelle Multiplikation Großer Zahlen. Computing **7**, 281–292 (1971)

Schöning, U.: A probabilistic algorithm for *k*-SAT and constraint satisfaction problems. Proceedings 40th Annual Symposium on Foundations of Computer Science, pp. 410–414. New York, USA (1999)

Schöning, U.: A probabilistic algorithm for *k*-SAT based on limited local search and restart. Algorithmica **32**, 615–623 (2002) (An earlier version appeared in 40th Annual Symposium on Foundations of Computer Science (FOCS '99), pp. 410–414)

Schoning, U., Pruim, R.: Gems of Theoretical Computer Science. Springer (1998)

Schrage, L.: A proof of the optimality of the shortest remaining processing time discipline. Oper. Res. **16**(1), 687–690 (1968)

Schrijver, A.: Combinatorial Optimization: Polyhedra and Efficiency. Springer, Berlin (2003)

Schrijver, A.: Theory of Linear and Integer Programming. Wiley, New York (1986)

Schuierer, S.: A lower bound for randomized searching on *m* rays. In: Computer Science in Perspective, pp. 264–277 (2003)

Schuierer, S.: Lower bounds in on-line geometric searching. Comput. Geom. **18**, 37–53 (2001)

Schuler, R.: An algorithm for the satisfiability problem of formulas in conjunctive normal form. J. Algorithms **54**(1), 40–44 (2005)

Schulman, L.J., Mor, T., Weinstein, Y.: Physical limits of heat-bath algorithmic cooling. Phys. Rev. Lett. **94**, 120501, pp. 1–4 (2005)

Schulman, L.J., Mor, T., Weinstein, Y.: Physical limits of heat-bath algorithmic cooling. SIAM J. Comput. **36**, 1729–1747 (2007)

Schulman, L.J., Vazirani, U.: Molecular scale heat engines and scalable quantum computation. Proc. 31st ACM STOC, Symp. Theory of Computing,pp. 322–329 Atlanta, 01–04 May 1999

Schultes, D., Sanders, P.: Dynamic highway-node routing. In: 6th Workshop on Experimental Algorithms. LNCS, vol. 4525, pp. 66–79. Springer, Berlin (2007)

Schulz, F., Wagner, D., Weihe, K.: Dijkstra's Algorithm On-Line: An Empirical Case Study from Public Railroad Transport. ACM J. Exp. Algorithmics **5**(12), 1–23 (2000)

Schulz, F., Wagner, D., Weihe, K.: Dijkstra's algorithm on-line: an empirical case study from public railroad transport. In: Proc. 3rd Workshop on Algorithm Engineering (WAE'99), pp. 110–123. Notes in Computer Science 1668. London, UK (1999)

Schulz, F., Wagner, D., Zaroliagis, C.: Using Multi-Level Graphs for Timetable Information in Railway Systems. In: Algorithm Engineering and Experiments – ALENEX 2002. Lect. Note Comp. Sci. **2409**, 43–59 (2002)

Schummer, J., Vohra, R.V.: Strategy-proof location on a network. J. Econ. Theor. **104**, 405–428 (2002)

Scott, A., Sorkin, G.: Faster Algorithms for MAX CUT and MAX CSP, with Polynomial Expected Time for Sparse Instances. In: Proceedings of RANDOM-APPROX 2003. LNCS, vol. 2764, pp. 382–395. Springer, Berlin (2003)

Scott, J., Ideker, T., Karp, R.M., Sharan, R.: Efficient Algorithms for Detecting Signaling Pathways in Protein Interaction Networks. J. Comput. Biol. **13**(2), 133–144 (2006)

Scott, S.: A study of stable marriage problems with ties. Ph. D. thesis, University of Glasgow, Department of Computing Science (2005)

Sedgewick, R.: Algorithms in Java, Parts 1–4, 3rd edn. Addison-Wesley, (2003)

Sedgewick, R., Bentley, J.: Fast algorithms for sorting and searching strings. In: Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '97), ACM, ed., pp. 360–369. ACM Press, New Orleans (1997)

Segall, A.: Distributed network protocols. IEEE Trans. Inform. Theory **29**, 23–35 (1983)

Seidel, R.: On the all-pairs-shortest-path problem. In: Proc. 24th ACM STOC pp. 745–749. Association for Computing Machinery, New York, USA (1992) Also JCSS **51**, 400–403 (1995)

Seiden, S.S.: On the online bin packing problem. J. ACM **49**, 640–671 (2002)

Sellers, P.: The theory and computation of evolutionary distances: pattern recognition. J. Algorithms **1**, 359–373 (1980)

Sellers, P.H.: On the Theory and Computation of Evolutionary Distances. SIAM J. Appl. Math. **26**, 787–793 (1974)

Sellis, T., Roussopoulos, N., Faloutsos, C.: The R^+-tree: A dynamic index for multi-dimensional objects. In: Proc. International Conference on Very Large Databases, 1987, pp. 507–518

Sen, A., Huson, M. L.: A New Model for Scheduling Packet Radio Networks. Proc. 15th Annual Joint Conference of the IEEE Computer and Communication Societies (IEEE INFOCOM'96), pp. 1116–1124, San Francisco, 24–28 March, 1996

Sen, P., Venkatesh, S.: Lower bounds for predecessor searching in the cell probe model. arXiv:cs.CC/0309033. See also ICALP'01, CCC'03, 2003

Sen, S., Chatterjee, S.: Towards a theory of cache-efficient algorithms. In: Proc. of 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2000), pp. 829–838. Society for Industrial and Applied Mathematics (2000)

Sentovich, E.M., Singh, K.J., Moon, C., Savoj, H., Brayton, R.K., Sangiovanni-Vincentelli, A.: Sequential Circuit Design using Synthesis and Optimization. In: Proc. of the IEEE International Conference on Computer Design: VLSI in Computers & Processors (ICCD), pp. 328–333. Cambridge, October 1992

Servedio, R.: On learning monotone DNF under product distributions. Inform Comput **193**(1), 57–74 (2004)

Servedio, R.A.: Smooth boosting and learning with malicious noise. JMLR **4**, 633–648 (2003)

Setubal, J.C., Meidanis, J.: Introduction to Computational Molecular Biology. PWS, Boston (1997)

Sevcik, K.C.: Scheduling for minimum total loss using service time distributions. J. ACM **21**, 66–75 (1974)

Seymour, P.D.: Packing directed circuits fractionally. Combinatorica **15**, 281–288 (1995)

Seymour, P.D., Thomas, R.: Call routing and the ratcatcher. Combinatorica **14**, 217–241 (1994)

Sgall, J.: On-line scheduling. In: Fiat, A., Woeginger, G.J. (eds.) Online Algorithms: The State of the Art, pp. 196–231. Springer (1998)

Shachnai, H., Tamir, T.: On two class-constrained versions of the multiple knapsack problem. Algorithmica **29**(3), 442–467 (2001)

Shachnai, H., Tamir, T.: Polynomial time approximation schemes for class-constrained packing problems. J. Sched. **4**(6) 313–338 (2001)

Shah, R., Varman, P.J., Vitter, J.S.: Online algorithms for prefetching and caching on parallel disks. In: Proceedings of the ACM Symposium on Parallel Algorithms and Architectures, pp. 255–264. ACM Press, New York (2004)

Shahrokhi, F., Matula, D.W.: The maximum concurrent flow problem. J. ACM **37**(2), 318–334 (1990)

Shalev-Shwartz, S., Singer, Y.: A new perspective on an old perceptron algorithm. In: Proceedings of the Eighteenth Annual Conference on Computational Learning Theory, (2005)

Shannon, C.: Presentation of a Maze Solving Machine, in Cybernetics, Circular, Causal and Feedback Machines in Biological and Social Systems. In: von Feerster, H., Mead, M., Teuber, H.L. (eds.) Trans. 8th Conf, New York, March 15–16, 1951. pp. 169–181. Josiah Mary Jr. Foundation, New York (1952)

Shannon, C.E.: A mathematical theory of communication. Bell Syst. Tech. J. **27**, 398–403 (1948)

Shannon, C.E.: A theorem on colouring lines of a network. J. Math. Phys. **28**, 148–151 (1949)

Shapley, L., Scarf, H.: On cores and indivisibility. J. Math. Econ. **1**, 23–37 (1974)

Shapley, S.L., Shubik, M.: The Assignment Game I: The Core. Int. J. Game. Theor. **1**, 111–130 (1971)

Sharan, R., Ideker, T.: Modeling cellular machinery through biological network comparison. Nat. Biotechnol. **24**, 427–433 (2006)

Shavit, N., Touitou, D.: Software transactional memory. Distrib. Comput., Special Issue **10**, 99–116 (1997)

Shawe-Taylor, J., Cristianini, N.: Kernel Methods for Pattern Analysis. Cambridge University Press, Cambridge. Book website: www.kernel-methods.net (2004)

Shenoy, N., Rudell, R.: Efficient implementation of retiming. In Proc. Intl. Conf. Computer-Aided Design, pp. 226–233. IEEE Press, Los Almitos (1994)

Shenvi, N., Kempe, J., Whaley, K.B.: A quantum random walk search algorithm. Phys. Rev. A **67**, 52–307 (2003)

Shepard, D.M., Ferris, M.C., Ove, R., Ma, L.: Inverse treatment planning for Gamma Knife radiosurgery. Med. Phys. **27**(12), 2748–2756 (2000)

Shi, W.: A Fast Algorithm for Area Minimization of Slicing Floorplan. In: IEEE Trans. Comput. Aided Des. **15**(12), 1525–1532 (1996)

Shibata, Y., Kida, T., Fukamachi, S., Takeda, M., Shinohara, A., Shinohara, T., Arikawa, S.: Speeding up pattern matching by text compression. In: Proc. 4th Italian Conference on Algorithms and Complexity (CIAC'00). LNCS, vol. 1767, pp. 306–315. Springer, Heidelberg (2000)

Shibata, Y., Matsumoto, T., Takeda, M., Shinohara, A., Arikawa, S.: A Boyer–Moore type algorithm for compressed pattern matching. In: Proc. 11th Annual Symposium on Combinatorial Pattern Matching (CPM'00). LNCS, vol. 1848, pp. 181–194. Springer, Heidelberg (2000)

Shigemizu, D., Maruyama, O.: Searching for regulatory elements of alternative splicing events using phylogenetic footprinting. In: Proceedings of the Fourth Workshop on Algorithms for Bioinformatics. Lecture Notes in Computer Science, pp. 147–158. Springer, Berlin (2004)

Shih, W.-K., Hsu, W.-L.: A new planarity test. Theor. Comput. Sci. **223**, pp. 179–191 (1999)

Shioura, A.: Fast Scaling Algorithms for M-convex Function Minimization with Application to the Resource Allocation Problem. Discret. Appl. Math. **134**, 303–316 (2004)

Shiple, T.R., Hojati, R., Sangiovanni-Vincentelli, A.L., Brayton, R.K.: Heuristic Minimization of BDDs Using Don't Cares. In: ACM Design Automation Conference, San Diego, CA, June (1994)

Shlomi, T., Segal, D., Ruppin, E., Sharan, R.: QPath: a method for querying pathways in a protein-protein interaction network. BMC Bioinform. **7**, 199 (2006)

Shmoys, D., Tardos, E.: An approximation algorithm for the generalized assignment problem. Math. Program. **62**(3A), 461–474 (1993)

Shmoys, D.B.: Approximation algorithms for facility location problems. In: Jansen, K., Khuller, S. (eds.) Approximation Algorithms for Combinatorial Optimization. Lecture Notes in Computer Science, vol. 1913, pp. 27–33. Springer, Berlin (2000)

Shmoys, D.B.: Cut problems and their application to divide-and-conquer. In: Hochbaum, D.S. (ed.) Approximation Algorithms for NP-hard Problems, pp. 192–235. PWS Publishing, Boston (1997)

Shmoys, D.B.: The design and analysis of approximation algorithms: Facility location as a case study. In: Thomas, R.R., Hosten, S., Lee, J. (eds) Proceedings of Symposia in Appl. Mathematics, vol. 61, pp. 85–97. AMS, Providence, RI, USA (2004)

Shmoys, D.B., Stein, C., Wein, J.: Improved Approximation Algorithms for Shop Scheduling Problems. SIAM J. Comput. **23**(3), 617–632 (1994)

Shmoys, D.B., Tardos, E., Aardal, K.: Approximation algorithms for facility location problems. In: Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC), pp. 265–274. ACM Press, New York (1997)

Shmoys, D.B., Wein, J., Williamson, D.P.: Scheduling parallel machines on-line. SIAM J. Comput. **24**, 1313–1331 (1995)

Shoikhet, K., Geiger, D.: A practical algorithm for finding optimal triangulations. In: Proc. National Conference on Artificial Intelligence (AAAI '97), pp. 185–190. Morgan Kaufmann, San Fransisco (1997)

Shor, P.: Algorithms for Quantum Computation: Discrete Logarithms and Factoring. In: Proceedings of the 35th Annual Symposium on Foundations of Computer Science, pp. 124–134, Santa Fe, 20–22 November 1994

Shor, P.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM J. Comput. **26**(5), 1484–1509 (1997)

Shor, P.W.: Fault-tolerant quantum computation. In: Proc. 37th Symp. on Foundations of Computer Science (FOCS) (1996). quant-ph/9605011

Shor, P.W.: Scheme for reducing decoherence in quantum computer memory. Phys. Rev. A **52**, R2493–R2496 (1995)

Shor, P.W., Preskill, J.: Simple proof of security of the BB84 quantum key distribution protocol. Phys. Rev. Lett. **85**, 441 (2000)

Shpilka, A., Wigderson, A.: Derandomizing homomorphism testing in general groups. In: Proceedings of the Thirty-Sixth Annual ACM Symposium on the Theory of Computing, pp. 427–435. ACM, NY, USA (2004)

Shwartz, A., Weiss, A.: Large Deviations for Performance Analysis. Chapman-Hall, Boca Raton (1994)

Shyu, J.M., Sangiovanni-Vincentelli, A.L., Fishburn, J.P., Dunlop, A.E.: Optimization-based Transistor Sizing. IEEE J. Solid. State. Circuits. **23**(2), 400–409 (1988)

Siek, J., Lee, L.Q., Lumsdaine, A.: The Boost Graph Library. Addison-Wesley, Cambridge (2002)

Siepel, A.C.: An algorithm to enumerate sorting reversals for signed permutations. J. Comput. Biol. **10**, 575–597 (2003)

Simon, D.: On the power of quantum computation. In: Proceedings of the 35th IEEE Symposium on the Foundations of Computer Science (FOCS), pp. 116–123, Santa Fe, 20–22 November 1994

Simon, D.R.: On the power of quantum computation. SIAM J. Comput. **26**(5), 1474–1483 (1997)

Singer-Cohen, K.B.: Random Intersection Graphs. Ph. D. thesis, John Hopkins University, Balimore (1995)

Singh, A.K., Anderson, J.H., Gouda, M.G.: The elusive atomic register. J. ACM **41**(2), 311–339 (1994) (Preliminary version in: Proc. 6th ACM Symp. Principles Distribt. Comput., 1987)

Singh, M., Lau, L.C.: Approximating minimum bounded degree spanning trees to within one of optimal. In: Proceedings of the thirty-ninth Annual ACM Symposium on Theory of Computing (STOC 2007), New York, NY, 2007, pp. 661–670

Singh, M., Prasanna, V.: Energy-Optimal and Energy-Balanced Sorting in a Single-Hop Wireless Sensor Network. In: Proc. First IEEE International Conference on Pervasive Computing and Communications (PerCom '03), pp. 302–317, Fort Worth, 23–26 March 2003

Singh, S., Raghavendra, C.S., Stepanek, J.: Power-Aware Broadcasting in Mobile Ad Hoc Networks. In: Proceedings of IEEE PIMRC'99, Osaka, September 1999

Singhal, M.: A taxonomy of distributed mutual exclusion. J. Parallel Distrib. Comput. **18**(1), 94–101 (1993)

Sinha, R., Zobel, J., Ring, D.: Cache-efficient string sorting using copying. ACM J. Exp. Algorithmics. **11** (2006)

Sipser, M.: A complexity theoretic approach to randomness. In: Proc. 15th Annual ACM Symposium on Theory of Computing, 1983, pp. 330–334

Sipser, M.: Introduction to the Theory of Computation, 2nd edn. Course Technology (2005)

Sipser, M., Spielman, D.: Expander codes. IEEE Trans. Inf. Theory **42**, 1710–1722 (1996)

Sitters, R.A., Stougie, L.: The generalized two-server problem. J. ACM **53**, 437–458 (2006)

Skjernaa, B.: Exact Algorithms for Variants of Satisfiability and Colouring Problems. Ph. D. thesis, University of Aarhus, Department of Computer Science (2004)

Skutella, M.: Convex quadratic and semidefinite relaxations in scheduling. J. ACM **46**(2), 206–242 (2001)

Skutella, M., Woeginger, G.J.: A PTAS for minimizing the weighted sum of job completion times on parallel machines. In: Proc. of 31st Annual ACM Symposium on Theory of Computing (STOC '99), pp. 400–407 (1999)

Slavik, P.: A tight analysis of the greedy algorithm for set cover. J. Algorithms **25**(2), 237–254 (1997)

Sleator, D.D., Tarjan, R.E.: A data structure for dynamic trees. J. Comput. Syst. Sci. **26**(3), 362–391 (1983)

Sleator, D.D., Tarjan, R.E.: Amortized efficiency of list update and paging rules. Commun. ACM **28**(2), 202–208 (1985)

Sleator, D.D., Tarjan, R.E.: Self-adjusting binary search trees. J. ACM **32**(3), 652–686 (1985)

Sleator, D.D., Tarjan, R.E., Thurston, W.P.: Rotation distance, triangulations, and hyperbolic geometry. In: Proceedings 18th ACM Symposium on Theory of Computing (STOC), Berkeley, 1986, pp. 122–135

Smith, D.R.: A new proof of the optimality of the shortest remaining processing time discipline. Oper. Res. **26**(1), 197–199 (1976)

Smith, J.M., Lee, D.T., Liebman, J.S.: An $O(N \log N)$ heuristic for Steiner minimal tree problems in the Euclidean metric. Networks **11**, 23–39 (1981)

Smith, T.F., Waterman, M.S.: Identification of Common Molecular Subsequences. J. Mol. Biol. **147**, 195–197 (1981)

Smith, W.E.: Various optimizers for single-stage production. Nav. Res. Log. Q. **3**, pp. 59–66 (1956)

Smyth, W.F.: Computing patterns in strings. Addison-Wesley, Boston, MA (2003)

Smyth, W.F.: Computing Patterns in Strings. Addison Wesley Longman, Harlow, UK (2002)

Smyth, W.F.: Repetitive perhaps, but certainly not boring. Theor. Comput. Sci. **249**(2), 343–355 (2000)

Sokol, D., Benson, G., Tojeira, J.: Tandem repeats over the edit distance. Bioinform. **23**(2), e30–e35 (2006)

Solis-Oba, R.: 2-approximation algorithm for finding a spanning tree with the maximum number of leaves. In: Proceedings of the 6th Annual European Symposium on Algorithms (ESA'98). Lecture Notes in Computer Science, vol. 1461, pp. 441–452. Springer, Berlin (1998)

Solymosi, T., Raghavan, T.E.S.: An Algorithm for Finding the Nucleolus of Assignment Games. Int. J. Game Theory **23**, 119–143 (1994)

Somenzi, F.: Colorado University Decision Diagram Package. http://vlsi.colorado.edu/~fabio/

Song, W.-Z., Wang, Y., Li, X.-Y. Frieder, O.: Localized algorithms for energy efficient topology in wireless ad hoc networks. In: ACM Int. Symposium on Mobile Ad-Hoc Networking and Computing (MobiHoc), Tokyo, 24–26 May 2004

Sørensen, O.W.: Polarization transfer experiments in high-resolution NMR spectroscopy. Prog. Nuc. Mag. Res. Spect. **21**, 503–569 (1989)

Spielman, D., Teng, S.-H.: Nearly-linear time algorithm for graph partitioning, graph sparsification, and solving linear systems. In: Proc. of the 36th Annual ACM Symp. on Theory of Computing, STOC'04, Chicago. USA, June 2004, pp. 81–90

Spielman, D.A., Teng, S.H.: Smoothed analysis of algorithms and heuristics: progress and open questions. In: Pardo, L.M., Pinkus, A., Süli, E., Todd, M.J. (eds.) Foundations of Computational Mathematics, pp. 274–342. Cambridge University Press, Cambridge, UK (2006)

Spillner, A.: A faster algorithm for the minimum weight triangulation problem with few inner points. In: Broersma, H., Johnson, H., Szeider, S. (eds.) Proceedings of the 1st ACiD Workshop. Texts in Algorithmics, vol. 4, pp. 135–146. King's College, London (2005)

Spillner, A.: Optimal convex partitions of point sets with few inner points. In: Proceedings of the 17th Canadian Conference on Computational Geometry (CCCG), 2005, pp. 34–37

Spirakis, P.: PRAM models and fundamental parallel algorithm techniques: Part II. In: Gibbons, A., Spirakis, P. (eds.) Lectures on Parallel Computation, pp. 41–66. Cambrige University Press, New York (1993)

Spirakis, P., Tsakalidis, A.: A Very Fast, Practical Algorithm for Finding a Negative Cycle in a Digraph. In Proc. of 13th ICALP, pp. 397–406 (1986)

Spirakis, P.G. Raptopoulos, C.: Simple and Efficient Greedy Algorithms for Hamilton Cycles in Random Intersection Graphs. In: Proc. of the 16th ISAAC. LNCS, vol. 3827, pp. 493–504. Springer, Berlin/Heidelberg (2005)

Spring, N., Mahajan, R., Wetherall, D.: Measuring ISP topologies with Rocketfuel. In: Proceedings of the ACM SIGCOMM'02 Conference. ACM, New York (2002)

Srikanth, T.K., Toueg, S.: Simulating Authenticated Broadcasts to Derive Simple Fault-Tolerant Algorithms. Distrib. Comp. **2**(2), 80–94 (1987)

Srinivasan, A.: Improved approximations for edge-disjoint paths, unsplittable flow, and related routing problems. Proc. IEEE FOCS, 1997, pp. 416–425

Srinivasan, A.: Improved approximations of packing and covering problems. In: Proceedings of the 27th Annual ACM Symposium on Theory of Computing, pp. 268–276 (1995)

Srinivasan, A., Teo, C.P.: A Constant-Factor Approximation Algorithm for Packet Routing and Balancing Local vs. Global Criteria. SIAM J. Comput. **30**(6), 2051–2068 (2000)

Stark, D.: The Vertex Degree Distribution of Random Intersection Graphs. Random Struct. Algorithms **24**, 249–258 (2004)

Stasko, J.: Animating Algorithms with X-TANGO. SIGACT News **23**, 67–71 (1992)

Stasko, J., Domingue, J., Brown, M., Price B.: Software Visualization: Programming as a Multimedia Experience. MIT Press, Cambridge, MA (1997)

Stasko, J., Kraemer, E.: A Methodology for Building Application-Specific Visualizations of Parallel Programs. J. Parall. Distrib. Comp. **18**, 258–264 (1993)

Steane, A.: Error correcting codes in quantum theory. Phys. Rev. Lett. **77**, 793–797 (1996)

Steane, A.: Multiple-particle interference and quantum error correction. Proc. R. Soc. London A **452**, 2551–2577 (1996)

Steel, M., Warnow, T.: Kaikoura tree theorems: computing the maximum agreement subtree. Inf. Process. Lett. **48**, 77–82 (1993)

Steel, M.A.: Recovering a tree from the leaf colourations it generates under a Markov model. Appl. Math. Lett. **7**, 19–24 (1994)

Steel, M.A.: The complexity of reconstructing trees from qualitative characters and subtrees. J. Classification **9**, 91–116 (1992)

Steele, J.M.: Cost of sequential connection for points in space. Oper. Res. Lett. **8**, 137–142 (1989)

Steen, H., Mann, M.: The ABC's (and XYZ's) of peptide sequencing. Nat. Rev. Mol. Cell Biol. **5**(9), 699–711 (2004)

Stefankovic, D.: Fourier transforms in computer science. Masters thesis, TR-2002-03, University of Chicago (2002)

Stege, U.: Resolving conflicts from problems in computational biology. Ph. D. Thesis, ETH Zürich, Institute of Scientific Computing (2000)

Stein, S.K.: Two combinatorial covering theorems. J. Comb. Theor. A **16**, 391–397 (1974)

Steinhaus, H.: Mathematical Snapshots. Oxford University Press, New York (1950)

Stepanec, G.F.: Basis systems of vector cycles with extremal properties in graphs. Uspekhi Mat. Nauk **19**, 171–175 (1964). (In Russian)

Stepanov, A., Lee, M.: The Standard Template Library. In: Technical Report X3J16/94–0095, WG21/N0482, ISO Programming Language C++ Project. Hewlett-Packard, Palo Alto CA (1994)

Stewart, J.W.: *BGP4, Inter-Domain Routing in the Internet*. Addison Wesley, Massacuhsetts (1998)

Stinson, D.R.: Cryptography: Theory and Practice, CRC Press, Inc (1995)

Stockmeyer, L.: Optimal Orientations of Cells in Slicing Floorplan Designs. Inf. Control **59**, 91–101 (1983)

Stockmeyer, L.J.: On approximation algorithms for #P. SIAM J. Comput. **14**, 849–861 (1985)

Stoer, M.: Design of Survivable Networks. Springer, Berlin (1992)

Stoica, I., Adkins, D., Zhuang, S., Shenker, S., Surana, S.: Internet Indirection Infrastructure. In: Proceedings of ACM SIGCOMM, pp. 73–88 (2002)

Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. In: Proceedings of the SIGCOMM 2001

Stojanovic, N., Dewar, K.: Identifying multiple alignment regions satisfying simple formulas and patterns. Bioinformatics **20**, 2140–2142 (2005)

Stojanovic, N., Florea, L., Riemer, C., Gumucio, D., Slighton, J., Goodman, M., Miller, W., Hardison, R.: Comparison of five methods for finding conserved sequences in multiple alignments of gene regulatory regions. Nucl. Acid. Res. **19**, 3899–3910 (1999)

Stojmenovic, I., Lin, X.: Loop-free hybrid single-path/flooding routing algorithms with guaranteed delivery for wireless networks. IEEE Trans. Paral. Distrib. Syst. **12**, 1023–1032 (2001)

Stok, L., Tiwari, V.: Technology Mapping. In: Hassoun, S., Sasou, T. (eds.) Logic Synthesis and Verification, pp. 115–139. Kluwer International Series In Engineering And Coputer Science Series. Kluwer Academic Publisher, Norwell (2002)

Storer, J.A.: Lossless image compression using generalized LZ1-type methods. In: Proceedings of Data Compression Conference, 1996, pp. 290–299

Stormo, G.: Consensus patterns in DNA. In: Doolittle, R.F. (ed.) Molecular evolution: computer analysis of protein and nucleic acid sequences. Methods in Enzymology, vol. 183, pp. 211–221 (1990)

Stormo, G., Hartzell III, G.W.: Identifying protein-binding sites from unaligned DNA fragments. Proc. Natl. Acad. Sci. USA. **88**, 5699–5703 (1991)

Strang, G.: Linear algebra and its applications, 2nd edn. Academic Press [Harcourt Brace Jovanovich Publishers], New York (1980)

STXXL: C++ Standard Library for Extra Large Data Sets. http://stxxl.sourceforge.net. Acessed: 15 March 2008

Subramaniam, S., Pope, S.B.: A mixing model for turbulent reactive flows based on euclidean minimum spanning trees. Combust. Flame **115**(4), 487–514 (1998)

Sudan, M.: Decoding of Reed–Solomon codes beyond the error-correction bound. J. Complex. **13**(1), 180–193 (1997)

Sudan, M.: List decoding: Algorithms and applications. SIGACT News. **31**(1), 16–27 (2000)

Suderman, M.: Layered Graph Drawing. Ph. D. thesis, McGill University, Montréal (2005)

Suderman, M., Whitesides, S.: Experiments with the fixed-parameter approach for two-layer planarization. J. Graph Algorithms Appl. **9**(1), 149–163 (2005)

Sugihara, K., Iri, M., Inagaki, H., Imai, T.: Topology-oriented implementation—an approach to robust geometric algorithms. Algorithmica **27**, 5–20 (2000)

Sugiyama, K., Tagawa, S., Toda, M.: Methods for visual understanding of hierarchical system structures. IEEE Trans. Syst. Man Cybernet. **11**(2), 109–125 (1981)

Sundar, R.: On the deque conjecture for the splay algorithm. Combinatorica **12**(1), 95–124 (1992)

Sundar, R.: Twists, turns, cascades, deque conjecture, and scanning theorem. In: Proceedings 30th IEEE Symposium on Foundations of Computer Science (FOCS), pp. 555–559 (1989)

Sundararajan, V., Parhi, K.: Low Power Synthesis of Dual Threshold Voltage CMOS VLSI Circuits. In: Proceedings of the International Symposium on Low Power Electronics and Design. pp. 139-144 (1999)

Sundararajan, V., Sapatnekar, S.S., Parhi, K.K.: Fast and exact transistor sizing based on iterative relaxation. Computer-Aided Design of Integrated Circuits and Systems, IEEE Trans. **21**(5),568–581 (2002)

Sutou, A., Dai, Y.: Global optimization approach to unequal sphere packing problems in 3D. J. Optim. Theor. Appl. **114**(3), 671–694 (2002)

Sutton, R.: Learning to predict by the methods of temporal differences. Mach. Learn. **3**, 9–44 (1988)

Sutton, R., Barto, A.: Reinforcement Learning. An Introduction. MIT Press, Cambridge (1998)

Svitkina, Z., Tardos, E.: Facility location with hierarchical facility costs. In: Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithm (SODA), pp. 153–161. SIAM, Philadelphia, PA, USA (2006)

Svitkina, Z., Tardos, É.: Min-Max multiway cut. In: 7th International workshop on Approximation algorithms for combinatorial optimization (APPROX), pp. 207–218, Cambridge, 2004 August 22–24

Swamy, C.: Correlation clustering: maximizing agreements via semidefinite programming. In: Proceedings of 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), New Orleans 2004, pp. 526–527

Swamy, C.: The effectiveness of stackelberg strategies and tolls for network congestion games. In: ACM-SIAM Symposium on Discrete Algorithms, Philadelphia, PA, USA (2007)

Swamy, C., Kumar, A.: Primal-dual algorithms for connected facility location problems. Algorithmica **40**(4), 245–269 (2004)

Swamy, C., Shmoys, D.B.: Fault-tolerant facility location. In: Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 735–736. SIAM, Philadelphia (2003)

Swofford, D.L., Olsen, G.J., Wadell, P.J., Hillis, D.M.: Phylogenetic inference. In: Hillis, D.M., Moritz, D.M., Mable, B.K. (eds.) Molecular systematics, 2nd edn. pp. 407–514. Sunderland, USA (1996)

Symvonis, A.: Routing on trees. Inf. Process. Lett. **57**(4), 215–223 (1996)

Szegedy, M.: On the quantum query complexity of detecting triangles in graphs. quant-ph/0310107

Szegedy, M.: Quantum speed-up of Markov chain based algorithms. In: Proceedings of the 45th IEEE Symposium on Foundations of Computer Science, pp. 32–41, Rome, Italy, 17–19 October 2004 (2004)

Szeider, S.: Minimal unsatisfiable formulas with bounded clause-variable difference are fixed-parameter tractable. J. Comput. Syst. Sci. **69**, 656–674 (2004)

Szeider, S.: On fixed-parameter tractable parameterizations of SAT. In: Giunchiglia, E., Tacchella, A. (eds.) Theory and Applications of Satisfiability, 6th International Conference, SAT 2003, Selected and Revised Papers. Lecture Notes in Computer Science, vol. 2919, pp. 188–202. Springer, Berlin (2004)

Ta-Shma, A.: Explicit one-probe storing schemes using universal extractors. Inf. Proc. Lett. **83**(5), 267–274 (2002)

Tabaska, J.E., Cary, R.B., Gabow, H.N., Stormo, G.D.: An RNA folding method capable of identifying pseudoknots and base triples. Bioinform. **14**, 691–699 (1998)

Takagi, H., Kleinrock, L.: Optimal Transmission Ranges for Randomly Distributed Packet Radio Terminals. IEEE Trans. Commun. **32**, 246–257 (1984)

Takahashi, H., Matsuyama, A.: An approximate solution for the Steiner problem in graphs. Math. Jap. **24**(6), 573–577 (1980)

Takaoka, T.: A new upper bound on the complexity of the all pairs shortest path problem. Inf. Proc. Lett. **43**, 195–199 (1992)

Takaoka, T.: Sub-cubic time algorithms for the all pairs shortest path problem. Algorithmica **20**, 309–318 (1998)

Tal, A., Dobkin, D.: Visualization of Geometric Algorithms. IEEE Trans. Visual. Comp. Graphics **1**, 194–204 (1995)

Talwar, K.: Bypassing the Embedding: Approximation Schemes and Compact Representations for Low Dimensional Metrics. In: Proceedings of the thirty-sixth Annual ACM Symposium on Theory of Computing (STOC'04), pp. 281–290 (2004)

Tamassia, R.: A dynamic data structure for planar graph embedding. 15th Int. Colloq. Automata, Languages, and Programming. LNCS, vol. 317, pp. 576–590. Springer, Berlin (1988)

Tamura, A.: Coordinatewise Domain Scaling Algorithm for M-convex Function Minimization. Math. Program. **102**, 339–354 (2005)

Tanenbaum, A.S.: Modern Operating Systems. Prentice-Hall, Englewood Cliffs (1992)

Tang, C.Y., Lu, C.L., Chang, M.D.T., Tsai, Y.T., Sun, Y.J., Chao, K.M., Chang, J.M., Chiou, Y.H., Wu, C.M., Chang, H.T., Chou, W.I.: Constrained multiple sequence alignment tool development and its application to RNase family alignment. In: Proc. of the First IEEE Computer Society Bioinformatics Conference (CSB 2002), 2002, pp. 127–137

Tang, X., Tian, R., Wong, M.D.F.: Optimal redistribution of white space for wirelength minimization. In: Tang, T.-A. (ed.) Proc. Asia South Pac. Design Autom. Conf., ACM Press, 18–21 Jan 2005, Shanghai. pp. 412–417 (2005)

Tannier, E., Bergeron, A., Sagot, M.-F.: Advances on Sorting by Reversals. Discret. Appl. Math. **155**, 881–888 (2006)

Tannier, E., Sagot, M.-F.: Sorting by reversals in subquadratic time. In: Proceedings of CPM'04. Lecture Notes Comput. Sci. **3109**, 1–13

Tarjan, R.: Sequential access in play trees takes linear time. Combinatorica **5**(4), 367–378 (1985)

Tarjan, R.E.: Data structures and network algorithms. In: CBMS-NSF Reg. Conf. Ser. Appl. Math., vol. 44. SIAM, Philadelphia (1983)

Tarjan, R.E.: Data Structures and Network Algorithms. SIAM, Philadelphia (1983)

Tarjan, R.E.: Dynamic trees as search trees via Euler tours, applied to the network simplex algorithm. Math. Prog. **78**, 169–177 (1997)

Tarjan, R.E., Vishkin, U.: An efficient parallel biconnectivity algorithm. SIAM. J. Comput. **14**, 862–874 (1985)

Tarjan, R.E., Werneck, R.F.: Dynamic trees in practice. In: Proceedings of the 6th Workshop on Experimental Algorithms (WEA). Lecture Notes in Computer Science, vol. 4525, pp. 80–93 (2007)

Tarjan, R.E., Werneck, R.F.: Self-adjusting top trees. In: Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 813–822 (2005)

Tassiulas, L., Sarkar, S.: Maxmin fair scheduling in wireless adhoc networks. IEEE J. Sel. Areas Commun. **23**(1), 163–173 (2005)

Tata, S., Hankins, R.A., Patel, J.M.: Practical suffix tree construction. In: Proc. 13th International Conference on Very Large Data Bases (VLDB), pp. 36–47, Toronto, Canada (2004)

Taubenfeld, G.: Synchronization algorithms and concurrent programming. Pearson Education – Prentice-Hall, Upper Saddle River (2006) ISBN: 0131972596

Taubenfeld, G.: The black-white bakery algorithm. In: 18th international symposium on distributed computing, October (2004). LNCS, vol. 3274, pp. 56–70. Springer, Berlin (2004)

Tenenbaum, J., de Silva, V., Langford, J.: A global geometric framework for nonlinear dimensionality reduction. Science **290**, 22 (2000)

Tesauro, G.J.: TD-gammon, a self-teaching backgammon program, achieves a master-level play. Neural Comput. **6**, 215–219 (1996)

Tesler, G.: Efficient algorithms for multichromosomal genome rearrangements. J. Comput. Syst. Sci. **63**(5), 587–609 (2002)

Tettelin, H., Radune, D., Kasif, S., Khouri, H., Salzberg, S.: Pipette Optimal Multiplexed PCR: Efficiently Closing Whole Genome Shotgun Sequencing Project. Genomics **62**, 500–507 (1999)

Thai, M.T., Wang F., Liu, D., Zhu, S., Du, D.-Z.: Connected Dominating Sets in Wireless Networks with Different Transmission Range. IEEE Trans. Mob. Comput. **6**(7), 721–730 (2007)

Thaker, D.D., Metodi, T.S., Cross, A.W., Chuang, I.L., Chong, F.T.: Quantum memory hierarchies: Efficient designs to match available parallelism in quantum computing. In: Proc. 33rd. Int. Symp. on Computer Architecture (ISCA), pp. 378–390 (2006) quant-ph/0604070

The Canterbury Tales Project: University of Birmingham, Brigham Young University, University of Münster, New York University, Virginia Tech, and Keio University. Website: http://www.canterburytalesproject.org/

The CGAL project homepage. http://www.cgal.org/. Accessed 6 Apr 2008

The CORE library homepage. http://www.cs.nyu.edu/exact/core/. Accessed 6 Apr 2008

The GMP webpage. http://gmplib.org/. Accessed 6 Apr 2008

The Stony Brook Algorithm Repository, http://www.cs.sunysb.edu/~algorith/. Accessed February 2008

Thiel, C.: On the complexity of some problems in algorithmic algebraic number theory, Ph. D. thesis. Universität des Saarlandes, Saarbrücken, Germany (1995)

Thimm, M.: On the approximability of the Steiner tree problem. Theor. Comput. Sci. **295**(1–3), 387–402 (2003)

Thomas, R.H.: A majority consensus approach to concurrency control for multiple copy databases. ACM Trans. Database Syst. **4**, 180–209 (1979)

Thompson, K.: Regular expression search algorithm. Commun. ACM **11**(6), 419–422 (1968)

Thorne, J.L., Kishino, H., Felsenstein, J.: An evolutionary model for maximum likelihood alignment of DNA sequences. J. Mol. Evol. **33**, 114–124 (1991)

Thorup, M.: Compact oracles for reachability and approximate distances in planar digraphs. In: Proc. 42nd IEEE Symposium on Foundations of Computer Science, 2001, pp. 242–251

Thorup, M.: Compact oracles for reachability and approximate distances in planar digraphs. J. ACM **51**, 993–1024 (2004)

Thorup, M.: Dynamic Graph Algorithms with Applications. In: Halldórsson, M.M. (ed) 7th Scandinavian Workshop on Algorithm Theory (SWAT), Norway, 5–7 July 2000, pp. 1–9

Thorup, M.: Equivalence between priority queues and sorting. In: Proc. 43rd FOCS, 2002, pp. 125–134

Thorup, M.: Faster deterministic sorting and priority queues in linear space. In: Proc. 9th SODA, 1998, pp. 550–555

Thorup, M.: Floats, integers, and single source shortest paths. J. Algorithms **35** (2000)

Thorup, M.: Fully-dynamic all-pairs shortest paths: Faster and allowing negative cycles. In: Proceedings of the 9th Scandinavian Workshop on Algorithm Theory (SWAT'04), pp. 384–396. Springer, Berlin (2004)

Thorup, M.: Integer priority queues with decrease key in constant time and the single source shortest paths problem. J. Comput. Syst. Sci. (special issue on STOC'03) **69**(3), 330–353 (2004)

Thorup, M.: Near-optimal fully-dynamic graph connectivity. In: Proc. 32nd ACM Symposium on Theory of Computing (STOC), 2000, pp. 343–350

Thorup, M.: On RAM priority queues. SIAM J. Comput. **30**(1), 86–109 (2000). Announced at SODA'96

Thorup, M.: Quick and good facility location. In: Proceedings 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 2003, pp. 178–185

Thorup, M.: Randomized sorting in $O(n \log \log n)$ time and linear space using addition, shift, and bit-wise boolean operations. J. Algorithms **42**(2), 205–230 (2002). Announced at SODA'97

Thorup, M.: Undirected single-source shortest paths with positive integer weights in linear time. J. ACM **46**(3), 362–394 (1999)

Thorup, M.: Worst-case update times for fully-dynamic all-pairs shortest paths. In: Proceedings of the 37th ACM Symposium on Theory of Computing (STOC 2005), ACM. New York (2005)

Thorup, M., Zwick, U.: Approximate distance oracles. In: Proceedings of the 33rd Annual ACM Symposium on the Theory of Computing, pp. 183–192. ACM Press, New York (2001)

Thorup, M., Zwick, U.: Approximate distance oracles. J. Assoc. Comput. Mach. **52**, 1–24 (2005)

Thorup, M., Zwick, U.: Spanners and emulators with sublinear distance errors. In: Proceedings of 17th Annual ACM-SIAM Symposium on Discrete Algorithms, 2006, pp. 802–809

Tinoco, I., Borer, P.N., Dengler, B., Levine, M.D., Uhlenbeck, O.C., Crothers, D.M., Gralla, J.: Improved estimation of secondary structure in ribonucleic acids. Nat. New Biol. **246**, 40–41 (1973)

Tinoco, I., Uhlenbeck, O.C., Levine, M.D.: Estimation of secondary structure in ribonucleic acids. Nature **230**, 362–367 (1971)

Tiwari, V., Ashar, P., Malik, S.: Technology Mapping for Low Power in Logic Synthesis. Integr. VLSI J. **20**(3), 243–268 (1996)

Toda, S., Watanabe, O.: Polynomial-Time 1-Turing Reductions from #**PH** to #**P**. Theor. Comput. Sci. **100**, 205–221 (1992)

Tompa, M.: Lecture notes. Department of Computer Science & Engineering, University of Washington. http://www.cs.washington.edu/education/courses/527/00wi/. (2000)

Toran, J.: On the hardness of graph isomorphism. SIAM J. Comput. **33**, 1093–1108 (2004)

Touati, H., Savoj, H., Lin, B., Brayton, R.K., Sangiovanni-Vincentelli, A.L.: Implicit State Enumeration of Finite State Machines using

BDDs. In: IEEE International Conference on Computer-Aided Design, pp. 130–133, November (1990)

Toueg, S., Perry, K.J., Srikanth, T.K.: Fast Distributed Agreement. SIAM J. Comput. **16**(3), 445–457 (1987)

Toussaint, G.T.: The relative neighborhood graph of a finite planar set. Pattern Recognit. **12**(4), 261–268 (1980)

TPIE — A Transparent Parallel I/O-Environment. http://www.cs.duke.edu/TPIE. Acessed: 15 March 2008

Traveling Salesman Problem. www.tsp.gatech.edu (2006). Accessed 28 Mar 2008

Treiber, R.: Systems programming: Coping with parallelism. Technical Report RJ5118, IBM Almaden Research Center (1986)

Trevisan, L.: Recycling queries in pcps and in linearity tests. In: Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, pp. 299–308. ACM, New York (1998)

Trevisan, L.: Some applications of coding theory in computational complexity. Quaderni Matematica **13**, 347–424 (2004) arXiv:cs.CC/0409044

Trevisan, L.: When Hamming meets Euclid: the approximability of geometric TSP and Steiner Tree. SIAM J. Comput. **30**(2), 475–485 (2000)

Trevisan, L., Sorkin, G., Sudan, M., Williamson, D.: Gadgets, approximation, and linear programming. SIAM J. Comput. **29**(6), 2074–2097 (2000)

Triantafillou, P., Ntarmos, N., Nikoletseas, S., Spirakis, P.: NanoPeer Networks and P2P Worlds. In:Proc. 3rd IEEE International Conference on Peer-to-Peer Computing (P2P 2003), pp. 40–46, Sept. 2003

Trick, M.: Michael Trick's coloring page: http://mat.gsia.cmu.edu/COLOR/color.html

Trimberger, S.: Field-Programmable Gate Array Technology. Springer, Boston, USA (1994)

Tromp, J.: How to construct an atomic variable. In: Proc. Workshop Distrib. Algorithms. Lecture Notes in Computer Science, vol. 392, pp. 292–302. Springer, Berlin (1989)

Trotter, W.T.: Current research problems: First Fit colorings of interval graphs. http://www.math.gatech.edu/~trotter/rprob.htm. Accessed 24 Dec 2007

Tsaknakis, H., Spirakis, P.: An Optimization Approach for Approximate Nash Equilibria. In: LNCS Proceedings of the 3rd International Workshop on Internet and Network Economics (WINE 2007), also in the Electronic Colloquium on Computational Complexity, (ECCC), TR07-067 (Revision), San Diego, 12–14 December 2007

Tsitsiklis, J.N., Van Roy, B.: Feature-based methods for large scale dynamic programming. Mach. Learn. **22**, 59–94 (1996)

Tullock, G.: Some problems of majority voting. J. Polit. Econ. **67**, 571–579 (1959)

Turpin, R., Coan, B.A.: Extending binary Byzantine Agreement to multivalued Byzantine Agreement. Inf. Process. Lett. **18**(2), 73–76 (1984)

Uemura, Y., Hasegawa, A., Kobayashi, S., Yokomori, T.: Tree adjoining grammars for RNA structure prediction. Theor. Comput. Sci. **210**, 277–303 (1999)

Ukkonen, E.: Finding approximate patterns in strings. J. Algorithms **6**, 132–137 (1985)

Ukkonen, E.: On-line construction of suffix trees. Algorithmica **14**, 249–260 (1995)

Ukkonen, E., Lemström, K., Mäkinen, V.: Sweepline the music! In: Klein, R. Six, H.W., Wegner, L. (eds.) Computer Science in Perspective, Essays Dedicated to Thomas Ottmann. LNCS, vol. 2598, pp. 330–342. Springer (2003)

Ullman, J.D.: NP-complete scheduling problems. J. Comput. Syst. Sci. **10**, 384–393 (1975)

Ullman, J.D.: The performance of a memory allocation algorithm. Tech. Rep. 100, Princeton University, Princeton, NJ (1971)

Umans, C., Villa, T., Sangiovanni-Vincentelli, A.L.: Complexity of two-level logic minimization. IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. **25**(7), 1230–1246 (2006)

Unger, W.: The complexity of the approximation of the bandwidth problem. In: 39th Annual Symposium on Foundations of Computer Science, IEEE, 8–11 Oct 1998, pp. 82–91.

Urrutia, J.: Routing with Guaranteed Delivery in Geometric and Wireless Networks. In: Stojmenovic, I. (ed.) Handbook of Wireless Networks and Mobile Computing, ch. 18 pp. 393–406. Wiley, Hoboken (2002)

Ursin, R., Jennewein, T., Aspelmeyer, M., Kaltenbaek, R., Lindenthal, M., Zeilinger, A.: Quantum teleportation link across the danube. Nature **430**(849), 849–849 (2004)

U.S. Census Bureau, Washington, DC: UA Census 2000 TIGER/Line Files. http://www.census.gov/geo/www/tiger/tigerua/ua_tgr2k.html (2002)

Vafeiadis, V., Herlihy, M., Hoare, T., Shapiro, M.: Proving correctness of highly-concurrent linearisable objects. In: PPoPP '06: Proceedings of the eleventh ACM SIGPLAN symposium on Principles and practice of parallel programming, pp. 129–136 (2006). doi: http://doi.acm.org/10.1145/1122971.1122992

Vaidya, P.: Solving linear equations with symmetric diagonally dominant matrices by constructing good preconditioners. Unpublished manuscript (1991)

Vaidya, P.M.: Minimum Spanning Trees in k-Dimensional Space. SIAM J. Comput. **17**(3), 572–582 (1988)

Valiant, L.: Learning disjunctions of conjunctions. In: Proc. 9th Int. Joint Conference on Artificial Intelligence, pp. 560–566, Los Angeles, August 1985

Valiant, L.G.: A scheme for fast parallel communication. SIAM J. Comput. **11**, 350–361 (1982)

Valiant, L.G.: A theory of the learnable. Commun. ACM **27**(11), 1134–1142 (1984)

Valiant, L.G., Brebner, G.: Universal schemes for parallel communication. In: Proceedings of the 13th ACM Symposium on Theory of Computing, 1981, pp. 263–277

van d. Heuvel, J., McGuiness, S.: Colouring the Square of a Planar Graph. CDAM Research Report Series, July 1999

van Emde Boas, P., Kaas, R., Zijlstra, E.: Design and implementation of an efficient priority queue. Math. Syst. Theor. **10**, 99–127 (1977). Announced by van Emde Boas alone at FOCS'75

van Santen, J.P.H., Buchsbaum, A.L.: Methods for optimal text selection. In: Proceedings of the European Conference on Speech Communication and Technology (Rhodos, Greece) **2**, 553–556 (1997)

van Vliet, A.: An improved lower bound for on-line bin packing algorithms. Inf. Proc. Lett. **43**, 277–284 (1992)

Vapnik, V. N.: Statistical Learning Theory. Wiley (1998)

Vapnik, V.: The Nature of Statistical Learning Theory. Springer, New York (1995)

Vapnik, V.N.: Estimations of dependences based on statistical data. Springer (1982)

Vapnik, V.N., Chervonenkis, A.Y.: On the uniform convergence of relative frequencies of events to their probabilities. Theory Probab. Apl. **16**, 264–280 (1971)

Varghese, G., Jayaram, M.: The Fault Span of Crash Failures. J. ACM **47**(2), 244–293 (2000)

Varian, H.R.: Economic mechanism design for computerized agents. In: Proceedings of the 1st Usenix Workshop on Electronic Commerce, 1995

Varian, H.R.: Position auctions. Int. J. Ind. Organ. **25**(6), 1163–1178 (2007) http://www.sims.berkeley.edu/~hal/Papers/2006/position.pdf. Accessed 29 March 2006

Vazirani, U.: Berkeley Lecture Notes. Fall 1997. Lecture 8. http://www.cs.berkeley.edu/~vazirani/qc.html (1997)

Vazirani, U., Vazirani, V.: Two-processor scheduling problem is in random NC. SIAM J. Comput. **18**(4), 1140–1148 (1989)

Vazirani, V.V.: A Theory of Alternating Paths and Blossoms for Proving Correctness of the $O(\sqrt{V}E)$ Maximum Matching Algorithm. Combinatorica **14**(1), 71–109 (1994)

Vazirani, V.V.: Approximation Algorithms. Springer, Berlin (2001)

Vazirani, V.V.: Approximation Algorithms. Springer, Berlin (2003)

Vempala, S.: Random projection: A new approach to VLSI layout. In: 39th Annual Symposium on Foundations of Computer Science, IEEE, 8–11 Oct 1998, pp. 389–398.

Vempala, S., Vetta, A.: Factor 4/3 approximations for minimum 2-connected subgraphs. In: Jansen, K., Khuller, S. (eds.) APPROX. Lecture Notes in Computer Science, vol. 1913, pp. 262–273. Springer, Berlin (2000)

Venkataraman, G., Sahni, S., Mukhopadhyaya, S.: A blocked all-pairs shortest paths algorithm. J. Exp. Algorithms **8** (2003)

Venter, J.C., Adams, M.D., Sutton, G.G., Kerlavage, A.R., Smith, H.O., Hunkapiller, M.: Shotgun sequencing of the human genome. Science **280**, 1540–1542 (1998)

Vetta, A.: Nash equilibria in competitive societies, with applications to facility location, traffic routing and auctions. In: 43rd Symp. on Foundations of Computer Science, pp. 416–425 (2002)

Vialette, S.: On the computational complexity of 2-interval pattern matching. Theor. Comput. Sci. **312**, 223–249 (2004)

Vickrey, W.: Counter speculation, auctions, and competitive sealed tenders. J. Financ. **16**, 8–37 (1961)

Vinh, L.S., von Haeseler, A.: Shortest triplet clustering: reconstructing large phylogenies using representative sets. BMC Bioinformatics **6**, 92 (2005)

Vishkin, U.: Optimal parallel pattern matching in strings. Proc. 12th ICALP, pp. 91–113 (1985)

Vitányi, P.M.B., Awerbuch, B.: Atomic shared register access by asynchronous hardware. In: Proc. 27th IEEE Symp. Found. Comput. Sci. pp. 233–243. Los Angeles, 27–29 October 1987. Errata, Proc. 28th IEEE Symp. Found. Comput. Sci., pp. 487–487. Los Angeles, 27–29 October 1987

Vitter, J. S., Shriver, E.A.M.: Algorithms for parallel memory. I: Two-level memories. Algorithmica. **12**(2/3), 110–147 (1994)

Vitter, J. S., Shriver, E.A.M.: Algorithms for parallel memory II: Hierarchical multilevel memories. Algorithmica **12**(2/3), 148–169 (1994)

Vitter, J.: Faster methods for random sampling. Commun. ACM **27**, 703–718 (1984)

Vitter, J.: Random sampling with a reservoir. ACM Trans. Math. Softw. **11**, 37–57 (1985)

Vitter, J.S.: External memory algorithms and data structures: Dealing with massive data. ACM Comput. Surv. **33**(2), 209–271 (2001)

Vitter, J.S.: Geometric and spatial data structures in external memory. In: Mehta, D., Sahni, S. (eds.) Handbook on Data Structures and Applications. CRC Press, Boca Raton (2005)

Vitter, J.S., Hutchinson, D.A.: Distribution sort with randomized cycling. J. ACM. **53** (2006)

Vizing, V.G.: On an estimate of the chromatic class of a p-graph (Russian). Diskret. Analiz. **3**, 25–30 (1964)

Vo, B.D., Vo, K.-P.: Compressing table data with column dependency. Theor. Comput. Sci. **387**, 273–283 (2007)

Vo, B.D., Vo, K.-P.: Using column dependency to compress tables. In: DCC: Data Compression Conference, pp. 92–101. IEEE Computer Society TCC, Washington DC, USA (2004)

Vo., K.-P.: Compression as data transformation. In: DCC: Data Compression Conference. IEEE Computer Society TCC, pp. 403. Washington DCD, USA (2006)

Vöcking, B.: Selfish load balancing. In: Nisan, N., Roughgarden, T., Tardos, É., Vazirani, V. (eds.) Algorithmic Game Theory. Cambridge University Press, New York, NY, USA (2007)

Vollmer, H.: Introduction to circuit complexity: a uniform approach. Springer, New York (1999)

von Neumann, J.: Probabilistic logic and the synthesis of reliable organisms from unreliable components. In: Shannon, C.E., McCarthy, J. (eds.) Automata Studies, pp. 43–98. Princeton University Press, Princeton (1956)

von Neumann, J., Morgenstern, O.: Theory of Games and Economic Behavior. Princeton University Press, Princeton, NJ (1944)

von Stackelberg, H.: Marktform und Gleichgewicht. Springer, Vienna (1934)

von zur Gathen, J., Gerhard, J.: Modern Comptuer Algebra, 2nd edn. Cambridge (2003)

Vovk, V.: Aggregating strategies. In: Fulk, M., Case, J. (eds.) Proceedings of the 3rd Annual Workshop on Computational Learning Theory, p. 371–383. Morgan Kaufmann, San Mateo (1990)

Vygen, J.: Approximation algorithms for facility location problems (lecture notes). Technical report No. 05950-OR, Research Institute for Discrete Mathematics, University of Bonn (2005) http://www.or.uni-bonn.de/~vygen/fl.pdf

Wagner, D., Willhalm, T., Zaroliagis, C.: Geometric Containers for Efficient Shortest Path Computation. ACM J. Exp. Algorithmics **10**(1.3), 1–30 (2005)

Wagner, R.A., Fischer, M.J.: The String-to-String correction Problem. J. ACM **21**(1), 168–173 (1974)

Wahlström, M.: An algorithm for the SAT problem for formulae of linear length. In: Proceedings of the 13th Annual European Symposium on Algorithms, ESA 2005. Lecture Notes in Computer Science, vol. 3669, pp. 107–118. Springer, Berlin (2005)

Wainwright, M., Jordan, M.: Variational inference in graphical models: the view from the marginal polytope. In: Proc. 41st Allerton Conf. on Communications, Control, and Computing, Monticello, October (2003)

Walras, L.: Elements of pure economics, or the theory of social wealth (1899, 4th ed; 1926, rev ed, 1954, Engl. Transl.) (1874)

Walter, J.E., Welch, J.L., Amato, N.M.: Distributed reconfiguration of metamorphic robot chains. J. Distrib. Comput. **17**(2), 171–189 (2004)

Wan, P.-J., Alzoubi, K.M., Frieder, O.: Distributed Construction of Connected Dominating Set in Wireless Ad Hoc Networks. In: IEEE INFOCOM 2002

Wan, P.-J., Calinescu, G., Li, X.-Y., Frieder, O.: Minimum-energy broadcast routing in static ad hoc wireless networks. ACM Wirel. Netw. Preliminary version appeared in IEEE INFOCOM (2000)**8**(6), 607–617 (2002)

Wan, P.-J., Calinescu, G., Yi, C.-W.: Minimum-power multicast routing in static ad hoc wireless networks. IEEE/ACM Trans. Netw. **12**, 507–514 (2004)

Wan, P.-J., Yi, C.-W.: Coverage by randomly deployed wireless sensor networks. In: Proceedings of the 4th IEEE International Symposium on Network Computing and Applications (NCA 2005), 27–29 July 2005

Wan, P.-J., Yi, C.-W.: On the longest edge of Gabriel graphs in wireless ad hoc networks. Trans. Parallel Distrib. Syst. **18**(1), 1–16 (2007)

Wan, P.-J., Yi, C.-W., Yao, F., Jia, X.: Asymptotic critical transmission radius for greedy forward routing in wireless ad hoc networks. In: Proceedings of the 7th ACM International Symposium on Mobile Ad Hoc Networking and Computing, 22–25 May 2006, pp. 25–36

Wang, C.C.: Multi-splay trees. Ph.D. Thesis, Carnegie Mellon University (2006)

Wang, C.C., Derryberry, J., Sleator, D.D.: $O(\log\log n)$-competitive dynamic binary search trees. In: Proc. 17th Annual ACM-SIAM Symposium on Discrete Algorithms, Miami, 2006, pp. 374–383

Wang, J.: Generating and solving 3-SAT, MSc Thesis. Rochester Institute of Technology, Rochester (2002)

Wang, J.: Medial axis and optimal locations for min-max sphere packing. J. Combin. Optim. **3**, 453–463 (1999)

Wang, J., Huang, M., Cheng, J.: A Lower Bound on Approximation Algorithms for the Closest Substring Problem. In: Proceedings COCOA 2007, vol. 4616 in LNCS, pp. 291–300 (2007)

Wang, L. Jiang, T.: On the complexity of multiple sequence alignment. J. Comp. Biol. **1**, 337–48 (1994)

Wang, L., Zhang, K., Zhang, L.: Perfect phylogenetic networks with recombination. J. Comput. Biol. **8**(1), 69–78 (2001)

Wang, T.C., Wong, D.F.: A Note on the Complexity of Stockmeyer's Floorplan Optimization Technique. In: Algorithmic Aspects of VLSI Layout, Lecture Notes Series on Computing, vol. 2, pp. 309–320 (1993)

Wang, T.C., Wong, D.F.: Optimal Floorplan Area Optimization. In: IEEE Trans. Comput. Aided Des. **11**(8), 992–1002 (1992)

Wang, W., Li, X.-Y.: Low-Cost routing in selfish and rational wireless ad hoc networks. IEEE Trans. Mobile Comput. **5**(5), 596–607 (2006)

Wang, W., Li, X.-Y., Chu, X.: Nash equilibria, dominant strategies in routing. In: Workshop for Internet and Network Economics (WINE). Lecture Notes in Computer Science, vol. 3828, pp 979–988. Springer, Hong Kong, China (2005)

Wang, W., Li, X.-Y., Sun, Z., Wang, Y.: Design multicast protocols for non-cooperative networks. In: Proceedings of the 24th IEEE INFOCOM. vol. 3, pp. 1596–1607, Miami, USA (2005)

Wang, W., Li, X.-Y., Wang, Y.: Truthful multicast in selfish wireless networks. In: Proceedings of the 10th ACM MOBICOM, pp. 245–259, Philadelphia, USA (2004)

Wang, Y., Li, X.-Y.: Efficient construction of bounded degree and planar spanner for wireless networks. In: ACM DIALM-POMC Joint Workshop on Foundations of Mobile Computing, San Diego, 19 September 2003

Wang, Y., Li, X.-Y.: Localized construction of bounded degree and planar spanner for wireless ad hoc networks, In: Proceedings of the 2003 joint workshop on Foundations of mobile computing (DIALM-POMC'03), 19 Sept 2003, pp. 59–68

Wang, Y., Wang, W., Li, X.-Y.: Efficient distributed low cost backbone formation for wireless networks. IEEE Trans. Parallel Distrib. Syst. **17**, 681–693 (2006)

Wang, Y., Wang, W., Li, X.-Y.: Efficient distributed low-cost backbone formation for wireless networks. In: Proceedings of 6th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc 2005), Urbana-Champaign, 25–27 May 2005

Warme, D.M., Winter, P., Zacharisen, M.: Exact algorithms for plane steiner tree problems: A computational study, Tech. Rep. DIKU-TR-98/11, Dept. of Computer Science, University of Copenhagen (1998)

Warme, D.M., Winter, P., Zacharisen, M.: GeoSteiner 3.1 package. ftp://ftp.diku.dk/diku/users/martinz/geosteiner-3.1.tar.gz. Accessed Oct. 2003

Warnow, T.: Some combinatorial optimization problems in phylogenetics. In: Lovász, L., Gyárfás, G., Katona, G., Recski, A., Székely, L. (eds.) Graph Theory and Combinatorial Biology. Bolyai Society Mathematical Studies, vol. 7, pp. 363–413. Bolyai János Matematikai Társulat (1999)

Warnow, T., Ringe, D., Taylor, A.: Reconstructing the evolutionary history of natural languages. In: Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'96), pp. 314–322 (1996)

Washietl, S., Hofacker, I.L., Stadler, P.F.: Fast and reliable prediction of noncoding RNA. Proc. Natl. Acad. Sci. USA **102**, 2454–59 (2005)

Waterman, M.S.: Efficient sequence alignment algorithms. J. Theor. Biol. **108**, 333–337 (1984)

Waterman, M.S.: Secondary structure of single-stranded nucleic acids. Adv. Math. Suppl. Stud. **1**, 167–212 (1978)

Waterman, M.S., Smith, T.F.: Rapid dynamic programming methods for RNA secondary structure. Adv. Appl. Math. **7**, 455–464 (1986)

Waterman, M.S., Smith, T.F., Singh, M., Beyer, W.A.: Additive evolutionary trees. J. Theor. Biol. **64**, 199–213 (1977)

Watkins, C.: Learning from Delayed Rewards. Ph. D. thesis, Cambridge University (1989)

Watkins, C., Dyan, P.: Q-learning. Mach. Learn. **8**(3/4), 279–292 (1992)

Watson, B.: Taxonomies and Toolkits of Regular Language Algorithms, Ph. D. Dissertation, Eindhoven University of Technology, The Netherlands (1995)

Wattenhofer, M., Wattenhofer, R., Widmayer, P.: Geometric Routing without Geometry. In: 12th Colloquium on Structural Information and Communication Complexity (SIROCCO), Le Mont Saint-Michel, France, May 2005

WEA. Beginning in 2001, the annual Workshop on Experimental and Efficient Algorithms is sponsored by EATCS. Workshop proceedings are published in the Springer LNCS series

Weber, R.R., Varaiya, P., Walrand, J.: Scheduling jobs with stochastically ordered processing times on parallel machines to minimize expected flow time. J. Appl. Probab. **23**, 841–847 (1986)

Weber, R.R., Weiss, G.: On an index policy for restless bandits. J. Appl. Probab. **27**, 637–648 (1990)

Wegener, I.: Branching Programs and Binary Decision Diagrams. SIAM (2000)

Wegman, M.N., Carter, J.L.: New hash functions and their use in authentication and set equality. *J. Comput. Syst. Sci.* **22**, 265–279 (1981)

Wei, Y.C., Cheng, C.K.: Towards Efficient Hierarchical Designs by Ratio Cut Partitioning. In: Proc. IEEE Int. Conf. on Computer-Aided Design, November 1989, pp. 298–301

Wein, R., Fogel, E., Zukerman, B., Halperin, D.: Advanced programming techniques applied to CGAL's arrangement package. Comput. Geom. Theor. Appl. **36**(1–2), 37–63 (2007)

Weiner, P.: Linear pattern matching algorithms. In: Proc. of the 14th Annual IEEE Symposium on Switching and Automata Theory, pp. 1–11. IEEE Press, New York (1973)

Weiss, A.: Personal Communication (1993)

Weiss, G. (ed.): Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. MIT Press, Cambridge, MA (1999)

Weiss, G.: Turnpike optimality of Smith's rule in parallel machine stochastic scheduling. Math. Oper. Res. **17**, 255–270 (1992)

Welch, T.A.: A technique for high-performance data compression. IEEE Comput. **17**, 8–19 (1984)

Werneck, R.F.: Design and Analysis of Data Structures for Dynamic Trees. Ph. D. thesis, Princeton University (2006)

Westbrook, J., Tarjan, R.E.: Maintaining bridge-connected and biconnected components on-line. Algorithmica **7**, 433–464 (1992)

Wexler, Y., Yakhini, Z., Kashi, Y., Geiger, D.: Finding approximate tandem repeats in genomic sequences. J. Comput. Biol. **12**(7), 928–42 (2005)

Whaley, R., Dongarra, J.: Automatically tuned linear algebra software (ATLAS). In: Proc. Supercomputing 98, Orlando, FL, November 1998. www.netlib.org/utk/people/JackDongarra/PAPERS/atlas-sc98.ps

Whittle, P.: Multiarmed bandit and the Gittins index. J. R. Stat. Soc. Series B **42**, 143–149 (1980)

Whittle, P.: Restless bandits: Activity allocation in a changing world. In: Gani, J. (ed.) A Celebration of Applied Probability. J Appl. Probab. **25A**, 287–298 (1988)

Wiberg, N.: Codes and Decoding on General Graphs, Ph. D. thesis, Linkoping University, Sweden (1996)

Wickremesinghe, R., Arge, L., Chase, J.S., Vitter, J.S.: Efficient sorting using registers and caches. ACM J. Exp. Algorithmics **7**, 9 (2002)

Wieselthier, J.E., Nguyen, G.D., Ephremides, A.: Energy-efficient broadcast and multicast trees in wireless networks. Mobile Netw. Appl. **7**, 481–492 (2002)

Wieselthier, J.E., Nguyen, G.D., Ephremides, A.: On the Construction of energy-Efficient Broadcast and Multicast Trees in Wireless Networks. IEEE Infocom **2**, 585–594 (2000)

Wiesmann, M., Schiper, A.: Comparison of database replication techniques based on total order broadcast. IEEE Trans. Knowl. Data Eng. **17**, 551–566 (2005)

Wiesner, S.: Conjugate coding. Sigact News **15**(1), 78–88 (1983)

Wigderson, A.: Improving the performance guarantee for approximate graph coloring. J. ACM **30**(4), 729–735 (1983)

Wilber, R.: Lower bounds for accessing binary search trees with rotations. SIAM J. Comput. **18**(1), 56–67 (1989)

Wile, B., Goss, J., Roesner, W.: Comprehensive Functional Verification. Morgan-Kaufmann (2005)

Willard, D.: Examining computational geometry, van Emde Boas trees, and hashing from the perspective of the fusion tree. SIAM J. Comput. **29**(3), 1030–1049 (2000). Announced at SODA'92

Williams, H.C.: Solving the Pell equation. In: Proc. Millennial Conference on Number Theory, pp. 397–435 (2002)

Williams, J.W.J.: Algorithm 232: Heapsort. Commun. ACM **7**(6), 347–348 (1964)

Williams, R.: A new algorithm for optimal 2-constraint satisfaction and its implications. Theor. Comput. Sci. **348**(2–3), 357–365 (2005)

Williams, R.: On Computing *k*-CNF Formula Properties. In: Theory and Applications of Satisfiability Testing. LNCS, vol. 2919, pp. 330–340. Springer, Berlin (2004)

Williams, R., Gomes, C., Selman, B.: On the connections between backdoors, restarts, and heavy-tailedness in combinatorial search, In: informal proceedings of SAT 2003 (Sixth International Conference on Theory and Applications of Satisfiability Testing, 5–8 May 2003, S. Margherita Ligure – Portofino, Italy), 2003, pp. 222–230

Williamson D.P., Goemans M.X., Mihail M., Vazirani V.V.: A Primal-Dual Approximation Algorithm for Generalized Steiner Network Problems. Combinatorica **15**(3), 435–454 (1995)

Wilson, L.B.: An analysis of the stable marriage assignment algorithm. BIT **12**, 569–575 (1972)

Wimer, S., Koren, I., Cederbaum, I.: Optimal Aspect Ratios of Building Blocks in VLSI. IEEE Trans. Comput. Aided Des. **8**(2), 139–145 (1989)

Win, S.: On a connection between the existence of *k*-trees and the toughness of a graph. Graphs Comb. **5**(1), 201–205 (1989)

Witten, E.: Quantum field theory and the Jones polynomial. Commun. Math. Phys. **121**(3), 351–399 (1989)

Witten, I., Moffat, A., Bell, I.: Managing Gigabytes, 2nd edn. Morgan Kaufmann (1999)

Witten, I.H., Bell, T.: The Calgary/Canterbury text compression corpus. Anonymous ftp from ftp://ftp.cpsc.ucalgary.ca:/pub/text.compression/corpus/text.compression.corpus.tar.Z

Witten, I.H., Moffat, A., Bell, T.C.: Managing Gigabytes: Compressing and Indexing Documents and Images, 2nd edn. Morgan Kaufmann, San Francisco, (1999)

Witten, I.H., Neal, R.M., Cleary, J.G.: Arithmetic coding for data compression. Commun. ACM **30**, 520–540 (1987)

Witwer, C., Hofacker, I.L., Stadler, P.F.: Prediction of consensus RNA secondary structures including pseudoknots. IEEE Trans. Comput. Biol. Bioinform. **1**, 66–77 (2004)

Wiuf, C.: Inference on recombination and block structure using unphased data. Genetics **166**(1), 537–545 (2004)

Wocjan, P., Yard, J.: The Jones polynomial: quantum algorithms and applications in quantum complexity theory. In: Quantum Information and Computation, vol. 8, no. 1 & 2, 147–180 (2008). arXiv.org:quant-ph/0603069 (2006)

Woeginger, G.J.: Exact algorithms for NP-hard problems: A survey. In: Combinatorial Optimization – Eureka, You Shrink. LNCS, vol. 2570, pp. 185–207. Springer, Berlin (2003)

Woeginger, G.J.: Space and time complexity of exact algorithms: some open problems. In: Proc. 1st Int. Workshop on Parameterized and Exact Computation (IWPEC 2004). LNCS, vol. 3162, pp. 281–290. Springer, Berlin (2004)

Wolsey, L.A.: An analysis of the greedy algorithm for the submodular set covering problem. Combinatorica **2**, 385–393 (1982)

Wong, C.H., Tam, Y.C.: Negative Cycle Detection Problem. In: Algorithms – ESA 2005. Lecture Notes in Computer Science, vol. 3669, pp. 652–663. Springer, Heidelberg (2005)

Wong, D.F., Liu, C.L.: A new algorithm for floorplan design. In: ACM/IEEE Design Automation Conference (DAC), November 1985, 23rd, pp. 101–107

Wong, D.F., Liu, C.L.: A New Algorithm for Floorplan Design. Proceedings of the 23rd ACM/IEEE Design Automation Conference, pp. 101–107 (1986)

Woodruff, D.: Lower Bounds for Additive Spanners, Emulators, and More. In: Proc. of Symp. on Foundations of Computer Science, Berckeley, Oct. 2006, pp. 389–398

Workman, C., Krogh, A.: No evidence that mRNAs have lower folding free energies than random sequences with the same dinucleotide distribution. Nucleic Acids Res. **27**, 4816–4822 (1999)

Wozencraft, J.M.: List Decoding. Quarterly Progress Report, Research Laboratory of Electronics. MIT **48**, 90–95 (1958)

Wu, B.Y., Chao, K.M.: Spanning Trees and Optimization Problems (Discrete Mathematics and Its Applications). Chapman Hall, USA (2004)

Wu, B.Y., K.-Chao, M., Tang, C.Y.: Approximation and exact algorithms for constructing minimum ultrametric trees from distance matrices. J. Combin. Optim. **3**, 199–211 (1999)

Wu, F.Y.: Knot Theory and statistical mechanics. Rev. Mod. Phys. **64**(4), 1099–1131 (1992)

Wu, J., Li, H.: A dominating-set-based routing scheme in ad hoc wireless networks. The special issue on Wirel. Netw. Telecommun. Systems J. **3**, 63–84 (2001)

Wu, Q.R.: Treatment planning optimization for Gamma unit radiosurgery. Ph. D. Thesis, The Mayo Graduate School (1996)

Wu, S., Manber, U.: A fast algorithm for multi-pattern searching. Report TR-94-17, Department of Computer Science, University of Arizona, Tucson, AZ (1994)

Wu, S., Manber, U.: Agrep – a fast approximate pattern-matching tool. In: Proceedings of USENIX Winter (1992) Technical Conference, pp. 153–162. USENIX Association, Berkeley (1992)

Wu, S., Manber, U.: Fast text searching allowing errors. Commun. ACM **35**(10), 83–91 (1992)

Wu, S., Manber, U., Myers, E.W.: A subquadratic algorithm for approximate regular expression matching. J. Algorithms **19**(3), 346–360 (1995)

Wu, W., Du, H., Jia, X., Li, Y., Huang, C.-H.: Minimum Connected Dominating Sets and Maximal Independent Sets in Unit Disk Graphs. Theor. Comput. Sci. **352**, 1–7 (2006)

Xu, C., Ma, B.: Software for Computational Peptide Identification from MS/MS. Drug Discov. Today **11**(13/14), 595–600 (2006)

Yan, M.: High Performance Algorithms for Phylogeny Reconstruction with Maximum Parsimony. Ph. D. thesis, Electrical and Computer Engineering Department, University of New Mexico, Albuquerque, January 2004

Yang, H., Wong, D.F.: Efficient Network Flow Based Min-Cut Balanced Partitioning. In: Proc. IEEE Int. Conf. on Computer-Aided Design, 1994, pp. 50–55

Yang, H.H., Wong, D.F.: Circuit clustering for delay minimization under area and pinconstraints. IEEE Trans. Comput.-Aided Des. Integr. Circ. Syst. **16**, 976–986 (1997)

Yannakakis, M.: Four pages are necessary and sufficient for planar graphs. In: Hartmanis, J. (ed.) Proceedings of the 18th Annual ACM-SIAM Symposium on Theory of Computing, pp. 104–108. ACM, New York (1986)

Yannakakis, M.: Graph-theoretic methods in database theory. In: Proc. 9-th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Nashville, 1990 pp. 230–242

Yao, A.: Near-optimal time-space tradeoff for element distinctness. SIAM J. Comput. **23**(5), 966–975 (1994)

Yao, A.: The complexity of pattern matching for a random string. SIAM J. Comput. **8**, 368–387 (1979)

Yao, A.C.-C.: On random 2–3 trees. Acta Inform. **9**, 159–170 (1978)

Yao, A.C.: New algorithms for bin packing. J. ACM **27**, 207–227 (1980)

Yao, A.C.: On Constructing Minimum Spanning Trees in k-Dimensional Spaces and Related Problems. SIAM J. Comput. **11**(4), 721–736 (1982)

Yao, A.C.C.: Should tables be sorted? J. Assoc. Comput. Mach. **28**(3), 615–628 (1981)

Yao, F., Demers, A., Shenker, S.: A Scheduling Model for Reduced CPU Energy, Proceedings of the 36th Annual IEEE Symposium on Foundations of Computer Science, pp. 374–382. IEEE Computer Society, Washington, DC, USA (1995)

Yap, C.K.: Theory of Real Computation according to EGC. To appear in LNCS Volume based on talks at a Dagstuhl Seminar "Reliable Implementation of Real Number Algorithms: Theory and Practice", Jan 8–13, (2006)

Yap, C.K., Dubé, T.: The exact computation paradigm. In: Du, D.Z., Hwang, F.K.: (eds.) Computing in Euclidean Geometry, 2nd edn., pp. 452–492. World Scientific Press, Singapore (1995)

Ye, Y.: A path to the Arrow-Debreu competitive market equilibrium, Math. Program. **111**(1–2), 315–348 (2008)

Ye, Y.: Exchange market equilibria with leontief's utility: freedom of pricing leads to rationality. WINE (2005)

Yi, C.-W., Wan, P.-J., Li, X.-Y., Frieder, O.: Asymptotic distribution of the number of isolated nodes in wireless ad hoc networks with Bernoulli nodes. In: IEEE Wireless Communications and Networking Conference (WCNC 2003), March 2003

Yi, C.-W., Wan, P.-J., Lin, K.-W., Huang, C.-H.: Asymptotic distribution of the Number of isolated nodes in wireless ad hoc networks with unreliable nodes and links. In: the 49th Annual IEEE GLOBECOM Technical Conference (GLOBECOM 2006), 27 Nov–1 Dec 2006

Yokoo, M.: The characterization of strategy/false-name proof combinatorial auction protocols: Price-oriented, rationing-free protocol. In: Proceedings of the 18th International Joint Conference on Artificial Intelligence, pp. 733–739 (2003)

Yokoo, M., Sakurai, Y., Matsubara, S.: Robust combinatorial auction protocol against false-name bids. Artif. Intell. **130**, 167–181 (2001)

Yokoo, M., Sakurai, Y., Matsubara, S.: Robust double auction protocol against false-name bids. Decis. Support. Syst. **39**, 23–39 (2005)

Yokoo, M., Sakurai, Y., Matsubara, S.: The effect of false-name bids in combinatorial auctions: New fraud in Internet auctions. Games Econ. Behav. **46**, 174–188 (2004)

Yoo-Ah Kim. Data migration to minimize the average completion time. J. Algorithms **55**, 42–57 (2005)

Young, N.E.: On-line file caching. Algorithmica **33**(3), 371–383 (2002)

Young, N.E.: On-Line Paging against Adversarially Biased Random Inputs. J. Algorithms **37**, 218 (2000)

Young, N.E.: Sequential and parallel algorithms for mixed packing and covering. In: Proceedings of 42nd Annual IEEE Symposium on Foundations of Computer Science (FOCS), 2001, pp. 538–546

Young, N.E.: The *k*-server dual and loose competitiveness for paging. Algorithmica **11**(6), 525–541 (1994)

Yu, L., Shih, H., Pfund, M., Carlyle, W., Fowler, J.: Scheduling of unrelated parallel machines: an application to PWB manufacturing. IIE Trans. **34**, 921–931 (2002)

Yu, Y., Prasanna, V.K.: Energy-Balanced Task Allocation for Collaborative Processing in Networked Embedded System. In: Proceedings of the 2003 Conference on Language, Compilers, and Tools for Embedded Systems (LCTES'03), pp. 265–274, San Diego, 11–13 June 2003

Yuan, J., Pixley, C., Aziz, A.: Constraint-Based Verfication. Springer (2006)

Yuan, Y.: Residence exchange wanted: a stable residence exchange problem. Eur. J. Oper. Res. **90**, 536–546 (1996)

Yun, H.S., Kim, J.: On Energy-Optimal Voltage Scheduling for Fixed-Priority Hard Real-Time Systems. ACM Trans. Embed. Comput. Syst. **2**, 393–430. ACM, New York, NY, USA (2003)

Yuster, R., Zwick, U.: Maximum Matching in Graphs with an Excluded Minor. In: Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA), 2007

Yuval, G.: An Algorithm for Finding All Shortest Paths Using $N^{2.81}$ Infinite-Precision Multiplications. Inf. Process. Lett. **4**(6), 155–156 (1976)

Zalka, C.: Grover's quantum searching algorithm is optimal. Phys. Rev. A **60**(4), 2746–2751 (1999)

Zarestkii, K.: Reconstructing a tree from the distances between its leaves. Uspehi Mathematicheskikh Nauk **20**, 90–92 (1965) (in russian)

Zaroliagis, C.D.: Implementations and experimental studies of dynamic graph algorithms. In: Experimental Algorithmics, Dagstuhl seminar, September 2000, Lecture Notes in Computer Science, vol. 2547. Springer (2002), Journal Article: J. Exp. Algorithmics 229–278 (2000)

Zelikovsky, A.Z.: The 11/6-approximation algorithm for the Steiner problem on networks. Algorithmica **9**, 463–470 (1993)

Zhang, H., Hou, J.: On deriving the upper bound of α-lifetime for large sensor networks. In: Proceedings of the 5th ACM International Symposium on Mobile Ad Hoc Networking & Computing (MobiHoc 2004), 24–26 March 2004

Zhang, J.: Approximating the two-level facility location problem via a quasi-greedy approach. In: Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 808–817. SIAM, Philadelphia (2004). Also, Math. Program. **108**, 159–176 (2006)

Zhang, J., Chen, B., Ye, Y.: A multiexchange local search algorithm for the capacitated facility location problem. Math. Oper. Res. **30**(2), 389–403 (2005)

Zhang, S., Xu, C., Deng, X.: Dynamic arbitrage-free asset pricing with proportional transaction costs. Math. Finance **12**(1), 89–97 (2002)

Zhang, Z., Berman, P., Wiehe, T., Miller, W.: Post-processing long pairwise alignments. Bioinformatics **15**, 1012–1019 (1999)

Zhao, B.Y., Huang, L., Stribling, J., Rhea, S.C., Joseph, A.D., Kubiatowicz, J.: Tapestry: A resilient global-scale overlay for service deployment. IEEE J. Sel. Areas Commun. (2003)

Zhao, J., Malmberg, R., Cai, L.: Rapid ab initio rna folding including pseudoknots via graph tree decomposition. In: Proc. Workshop on Algorithms in Bioinformatics. Lecture Notes in Computer Science, vol. 4175, pp. 262–273. Springer, Berlin (2006)

Zhao, W., Ammar, M., Zegura, E.: A message ferrying approach for data delivery in sparse mobile ad hoc networks. In: Murai, J., Perkins, C., Tassiulas, L. (eds.) 5th ACM international symposium on Mobile ad hoc networking and computing (MobiHoc 2004), pp 187–198. ACM Press, Roppongi Hills, Tokyo (2004)

Zheng, R., He, G., Gupta, I., Sha, L.: Time idexing in sensor networks. In: Proceedings of 1st IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS), Fort Lauderdale, 24–27 October 2004

Zheng, S.Q., Lim, J.S., Iyengar, S.S.: Finding obstacle-avoiding shortest paths using implicit connection graphs. IEEE Trans. Comput. Aided Des. **15**, 103–110 (1996)

Zhong, S., Li, L., Liu, Y., Yang, Y.R.: On designing incentive-compatible routing and forwarding protocols in wireless ad-hoc networks –an integrated approach using game theoretical and cryptographic techniques. In: Proceedings of the 11th ACM Annual international Conference on Mobile Computing and Networking, Cologne, 28 August – 2 September 2005

Zhou, H.: A new efficient retiming algorithm derived by formal manipulation. In: Workshop Notes of Intl. Workshop on Logic Synthesis, Temecula, CA, June (2004)

Zhou, H.: Deriving a new efficient algorithm for min-period retiming. In Asia and South Pacific Design Automation Conference, Shanghai, China, Jan. ACM Press, New York (2005)

Zhou, H., Shenoy, N., Nicholls, W.: Efficient minimum spanning tree construction without delaunay triangulation. In: Proc. Asian and South Pacific Design Automation Conference, Yokohama, Japan (2001)

Zhou, H., Shenoy, N., Nicholls, W.: Efficient spanning tree construction without delaunay triangulation. Inf. Proc. Lett. **81**, 271–276 (2002)

Zhou, L.: On a conjecture by Gale about one-sided matching problems. J. Econ. Theory **52**(1), 123–135 (1990)

Zhou, L., van Renesse, R., Marsh, M.: Implementing IPv6 as a Peer-to-Peer Overlay Network. In: Proceedings of the 21st IEEE Symposium on Reliable Distributed Systems (SRDS'02), pp. 347 (2002)

Zhu, D., Melhem, R., Childers, B.: Scheduling with Dynamic Voltage/Speed Adjustment Using Slack Reclamation in Multi-Processor RealTime Systems. Proceedings of the 22nd IEEE Real-Time Systems Symposium (RTSS'01), pp. 84–94. IEEE Computer Society, Washington, DC, USA (2001)

Zhuang, S.Q., Zhao, B.Y., Joseph, A.D., Katz, R.H., Kubiatowicz, J.: Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination. In: Proceedings of the Eleventh International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV 2001), 2001

Zibert, K., Saal, R.: On Computer Aided Hybrid Circuit Layout. Proceedings of the IEEE Intl. Symp. on Circuits and Systems, pp. 314–318 (1974)

Ziv, J.: Personal communication (1995)

Ziv, J., Lempel, A.: A universal algorithm for sequential data compression. IEEE Trans. Inf. Theor. **23**, 337–343 (1977)

Ziv, J., Lempel, A.: Compression of Individual Sequences via Variable Rate Coding. IEEE Trans. Inf. Theory **24**(5), 530–536 (1978)

Zollinger, A.: Networking Unleashed: Geographic Routing and Topology Control in Ad Hoc and Sensor Networks, Ph. D. thesis, ETH Zurich, Switzerland Diss. ETH 16025 (2005)

Zuckerman, D.: Linear degree extractors and the inapproximability of max clique and chromatic number. In: Proceedings of the 38th annual ACM symposium on Theory of Computing (2006) pp. 681–690.

Zuker, M.: Calculating nucleic acid secondary structure. Curr. Opin. Struct. Biol. **10**, 303–310 (2000)

Zuker, M.: On finding all suboptimal foldings of an RNA molecule. Science **244**, 48–52 (1989)

Zuker, M., Stiegler, P.: Optimal computer folding of large RNA sequences using thermodynamics and auxiliary information. Nucleic Acids Res. **9**, 133–148 (1981)

Zukowski, M., Heman, S., Boncz, P.A.: Architecture-conscious hashing. In: Proceedings of the International Workshop on Data

Management on New Hardware (DaMoN), Article No. 6. ACM Press, Chicago, Illinois, USA, 25 June 2006

Zwick, U.: All pairs shortest paths using bridging sets and rectangular matrix multiplication. J. ACM **49**(3), 289–317 (2002)

Zwick, U.: Exact and approximate distances in graphs – a survey. In: Proc. 9th European Symposium on Algorithms (ESA), 2001, pp. 33–48. See updated version at http://www.cs.tau.ac.il/~zwick/

Zyablov, V.V., Pinsker, M.S.: List cascade decoding. Probl. Inf. Trans. **17**(4), 29–34 (1981) (in Russian); pp. 236–240 (in English) (1982)

Zykov, A.A.: Theory of Finite Graphs. Nauka, Novosibirsk (1969). (In Russian)

Index